AN MS-DOS FILE SERVER FOR A PC-LAN

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of

MASTER OF TECHNOLOGY

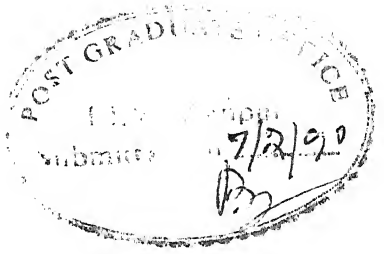
by

CIGY CYRIAC

to the

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

FEBRUARY, 1990



CERTIFICATE

Certified that this work entitled 'An MS-DOS file server for a PC-LAN' has been carried out under my supervision and has not been submitted elsewhere for a degree.

DR. A. JOSHI
Professor
Department of Electrical Engineering
Indian Institute of Technology, Kanpur

ACKNOWLEDGMENTS

I express my gratitude and respect to Dr. A. Joshi for his able guidance and encouragement during the period of my work.

I am also indebted to Dr. S. K. Bose and Dr. K. R. Srivathsan of the Electrical Engineering department, as well as Dr. Sadagopan of the IME department, for the useful discussions I had with them in the initial stages of my work.

I gratefully acknowledge the co-operation extended by Mr. S. S. Bhatnagar, Mr. N. R. Gogate, Mr. A. Roy and Mr. M. Sathish of the MDS Lab. Thanks are also due to my friends, especially Mr. Rakesh Kumar and Mr. S. Hemachandran.

Finally I would like to thank Dennis Ritchie of Bell Laboratories for having developed the powerful and versatile 'C' programming language, with which software development for this project was a pleasant and valuable experience.

CIBY CYRIAC

ABSTRACT

This thesis discusses the development of a file server for the IIT-K PC-LAN, a token ring local area network for low-cost interconnection of IBM-compatible personal computers. Individual PC users are provided with transparent access to the MS-DOS file system maintained by the server. To a user, the remote file system appears to be on a virtual drive on his local machine, and can be accessed as easily as a local drive. Users are provided with facilities for sharing one another's files on the server, while also being able to exercise control over how and to what extent this can be done. In addition to the actual server, the software developed includes a redirector module for each client PC, a network driver and a utility package.

CONTENTS

CHAPTER 1
INTRODUCTION

With computers becoming smaller and cheaper, users have become more interested in connecting them together to form networks. A Local Area Network (LAN), characterized by its high data rate and small area of coverage, connects together a collection of computers, terminals and peripherals located in the same building, or in adjacent buildings, allowing them to intercommunicate and exploit the advantages of distributed computing.

1.1 The user-server model

LANs increase the functionality of application software by making a whole range of resources and peripherals available to the user. Hardware such as large hard disks, printers, modems, etc. can be accessed by computer users on a network through properly integrated LAN software. This allows users to share data and expensive peripherals and send messages to one another. In this **user-server** model of distributed computing (Fig. 1.1) the user does all his actual computing on his personal computer, but various centrally located server machines on the network carry out specific functions on behalf of any user who requests them.

For example, the network could have several **disk servers** that read and write raw disk blocks in response to user requests. For a higher level of abstraction, it could provide **file servers**, offering file system services. Such a server could present an interface identical to UNIX, MS-DOS or any other popular file

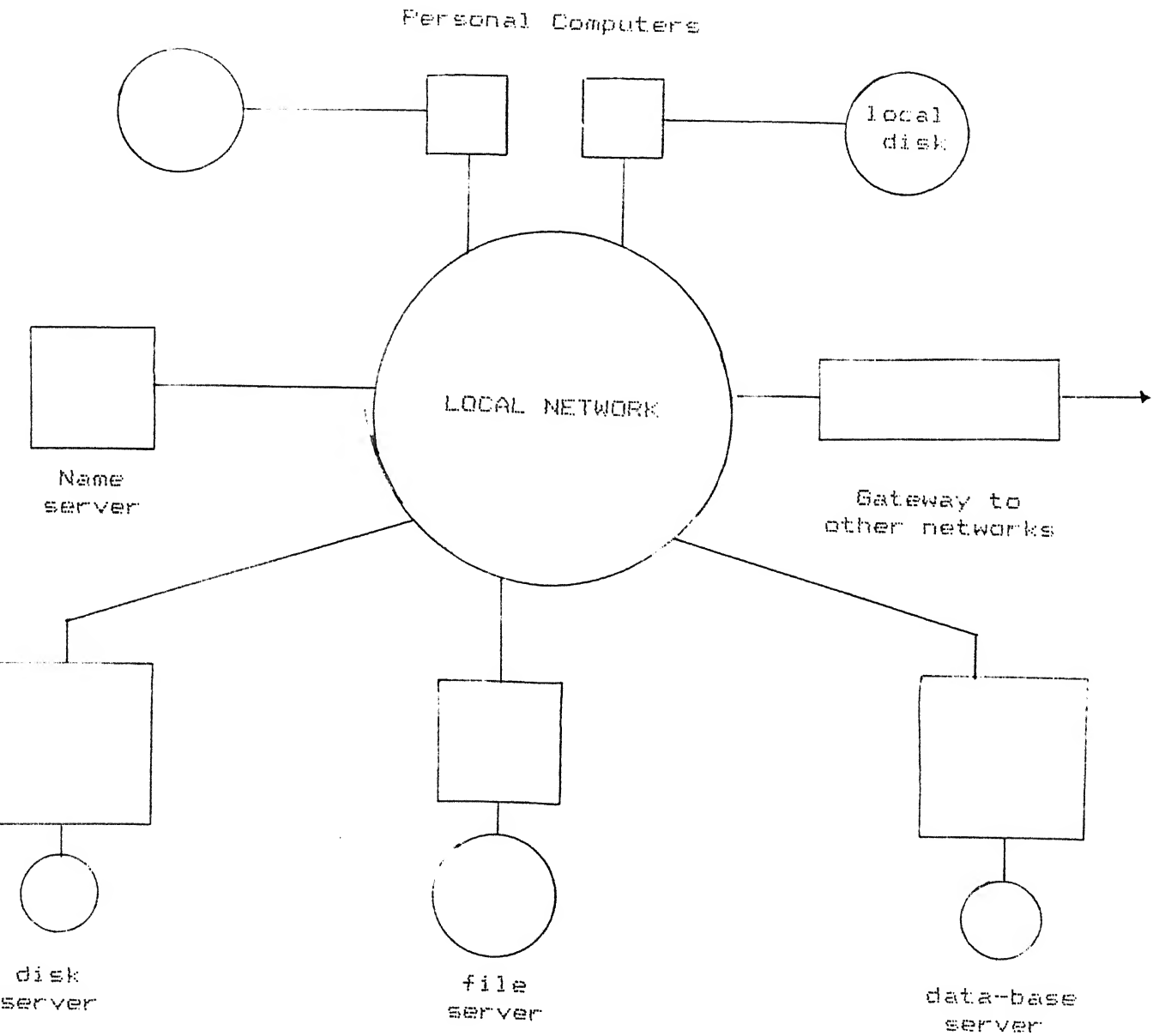


Fig. 1.1 The user-server model

system, allowing users to access remote files just as they would access files on their local secondary storage. Thus it would honor requests for opening, creating, reading, writing or deleting files, making or removing subdirectories, changing working directories, etc.

A valuable network facility is the ability to share files among multiple users. This means that more than one user can read or update a file. The owner of a file should be allowed to decide if and how it can be shared with other users. Systems that allow sharing also provide some sort of locking facility to prevent multiple users from trying to simultaneously update a file.

1.2 Objective of this project

Previous project activities at IIT-K have succeeded in developing a LAN for low-cost interconnection of PCs. The PC-LAN is based on the token ring topology and can support a maximum of 255 PCs. (Fig.1.2). Simple file transfer protocols like TFTP have already been implemented on it. More information on the PC-LAN can be obtained from Appendix A and from references [1] and [2].

This thesis describes the development and implementation of a file server for the PC-LAN, to allow individual PC users to transparently access files maintained in an MS-DOS file system on a large central secondary storage device, e.g. a 100 MB hard disk. Transparent access ensures that users can use standard application software and still access remote files just as easily as their local files, without caring about the difference and without having to learn too many extra commands and procedures to achieve it.

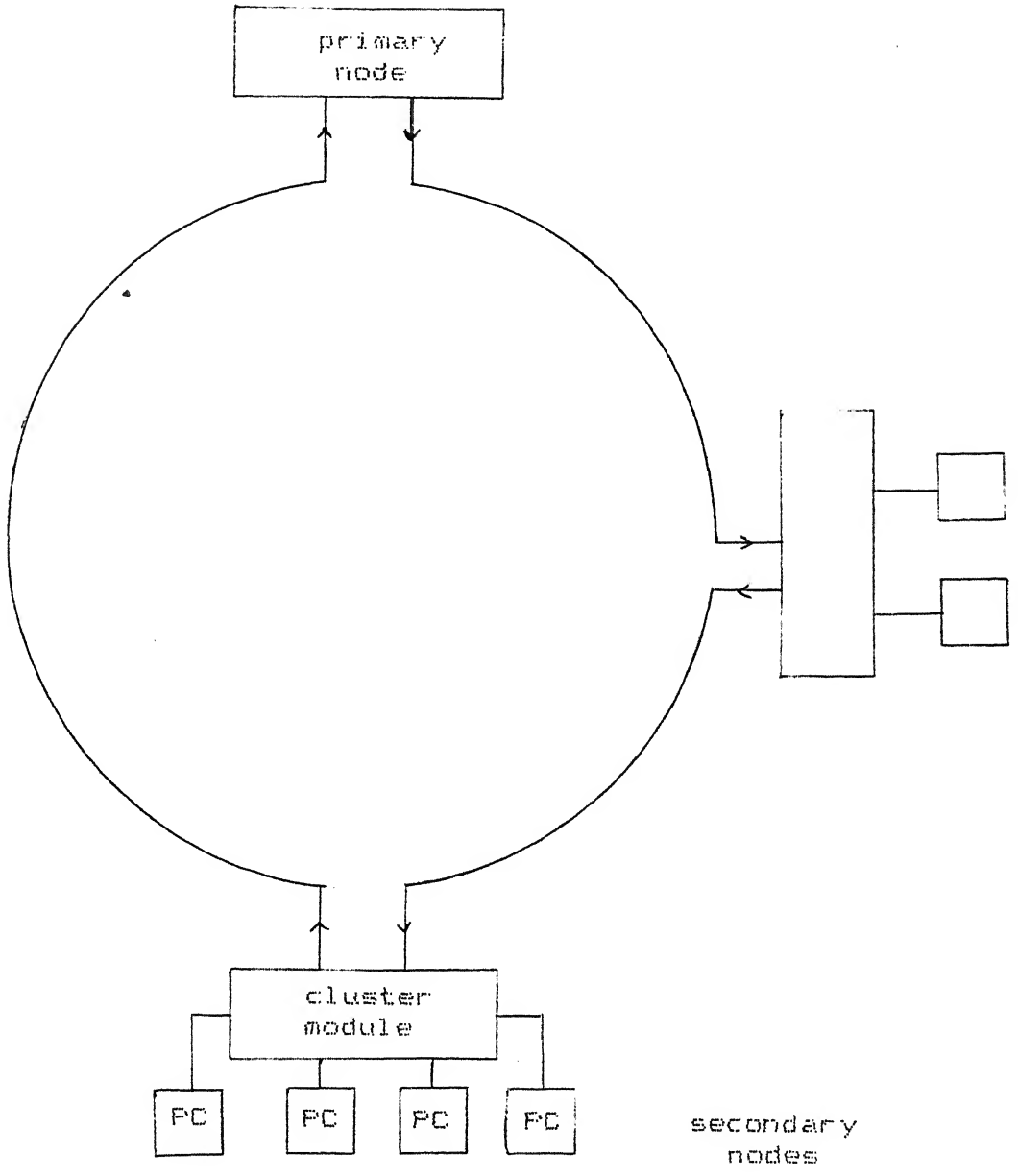


Fig 1.2 The PC-LAN topology

In addition the system is expected to support file sharing and locking, to allow users to make the best possible use of the network, while also guaranteeing security and protection against unauthorized access. The system should provide reliable and error-free service with a minimum amount of delay.

1.3 Overview of the system

The software developed consists of four main modules :

i. The **server** is a dedicated process running under MS-DOS on the PC meant to be used as the file server. It maintains an MS-DOS file system on its local hard disk(s), which is open for access by clients through the network. Individual users are assigned subdirectories in the root of the file system. The server accepts requests from clients for file/directory operations, processes them through calls to the local DOS after making sure that the user is permitted to do the requested operation in the specified directory, and sends back replies indicating success or failure.

ii. The **redirector** is a module that is memory-resident in all client PCs. It provides a transparent interface between user processes and the server by intercepting the process' DOS system calls. Its primary responsibility is to ensure that the user's requests for file/directory accesses are channeled to the proper point : the local DOS for local requests and the server for remote requests. It also takes care of all the local housekeeping associated with the accessed files and translates system call parameters between DOS and server formats.

iii. The **network driver** is a device driver forming the interface to the PC-LAN hardware and is used by both the redirector and the server to send and receive messages over the token ring. It frees these processes from having to know about the details of the LAN hardware used and the protocol followed for communication.

iv. The **utilities** are a collection of functions that allow users to login and logout conveniently, as well as to make use of some special functions supported by the server. These include calls to inspect or modify information like rights, group membership, etc.

Fig. 1.3.1 shows a typical layout of the system, indicating the locations of the modules mentioned above. Fig. 1.3.2 shows the flow of local and remote requests.

1.4 Organization of the thesis

Chapter 2 elaborates on the requirements outlined above and discusses the different design choices and issues. It also provides a general outline of the complete system. Chapter 3 looks at the details of the actual implementation and gives a module-wise description of the software. Chapter 4 reports on the performance of the system and suggests directions in which future work can be done.

Appendix A contains general details about the PC-LAN while Appendix B provides notes on using, maintaining and modifying the system. The complete source listing of the system is contained in Appendix C.

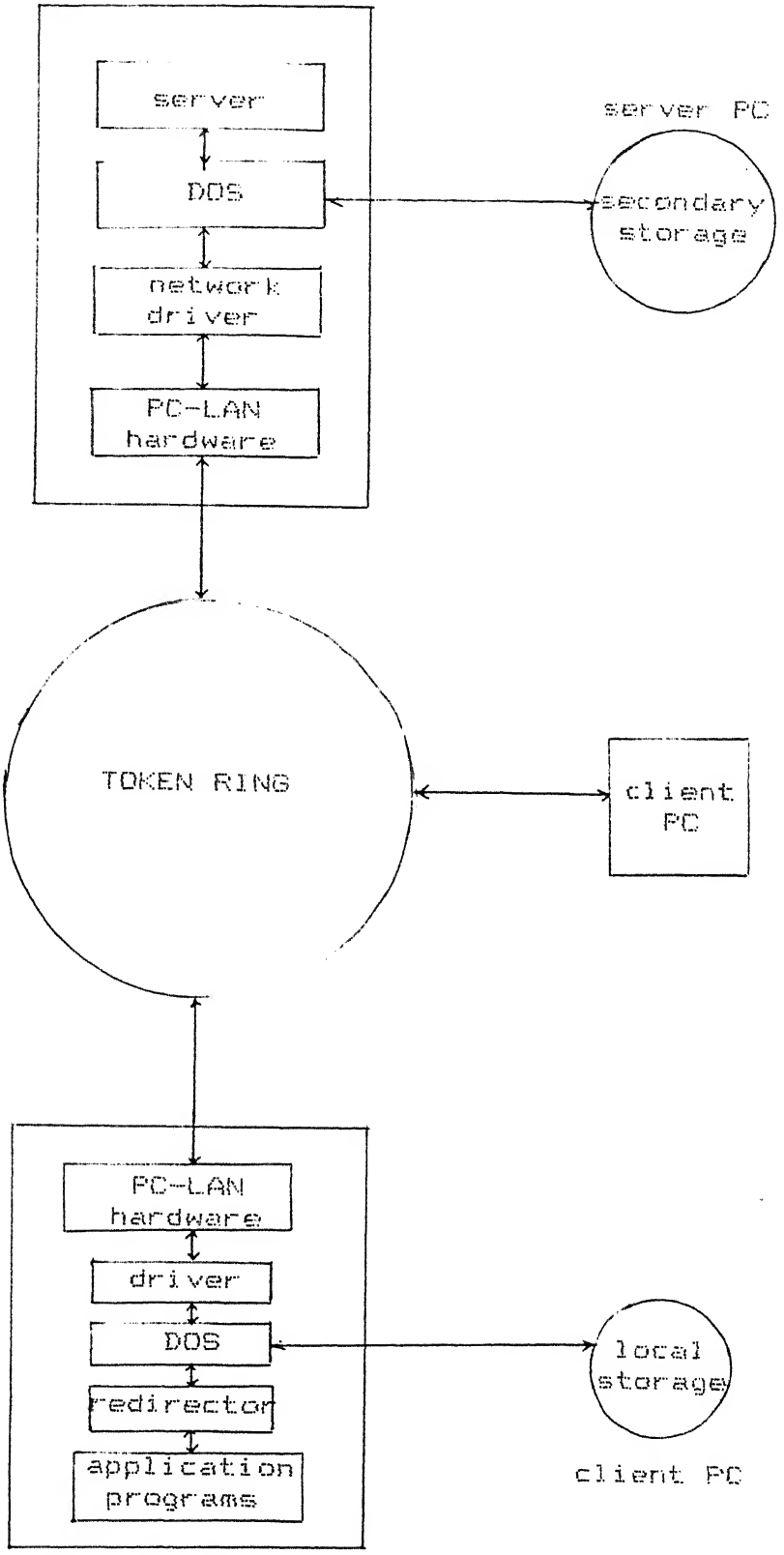


Fig. 1.3.1 Layout of the system

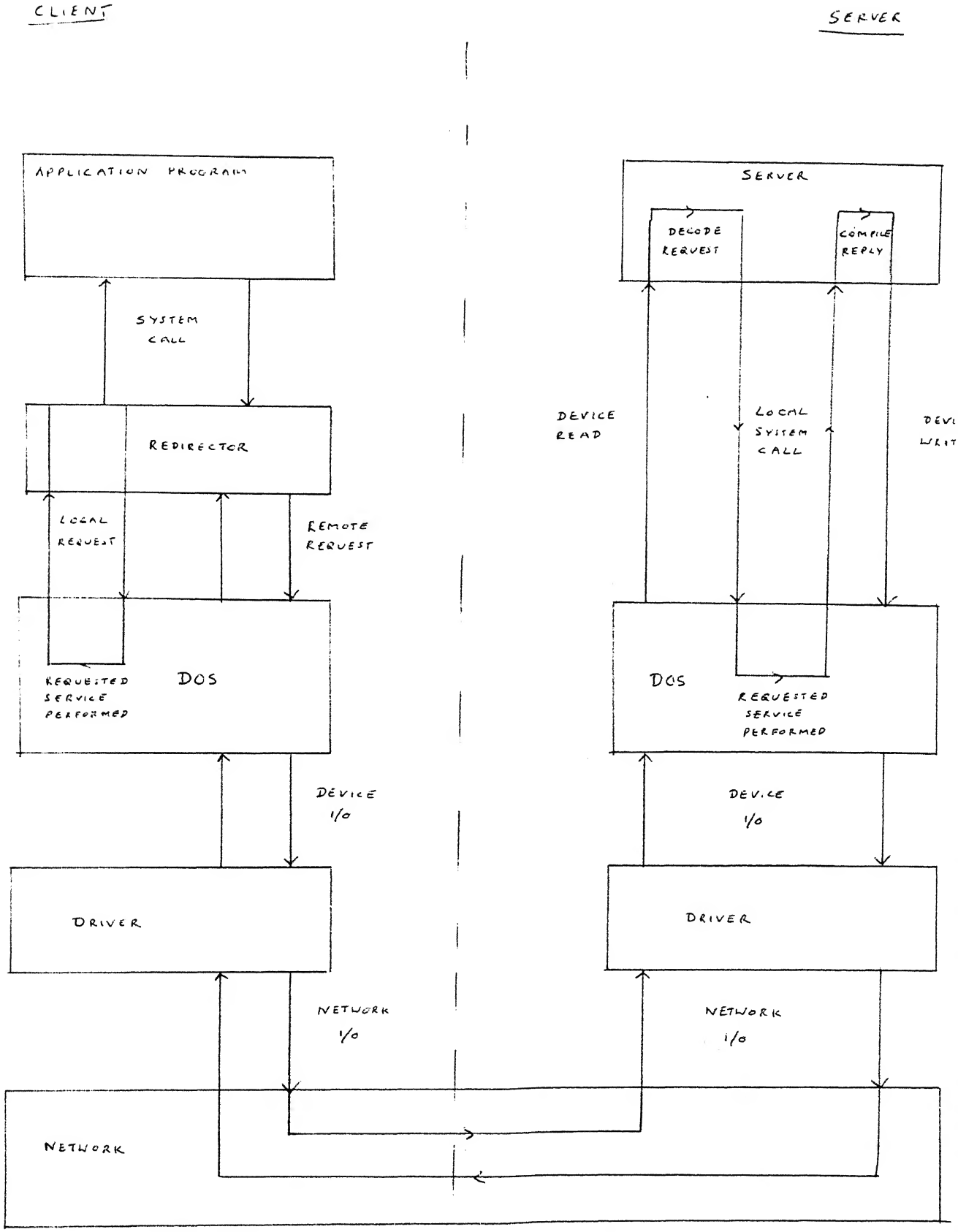


Fig. 1.3-2 FLOW OF LOCAL AND REMOTE REQUESTS

CHAPTER 2

DESIGN CONSIDERATIONS

2.1 The Server

2.1.1 Disk service vs. File service

There are two possible approaches to the problem of providing remote secondary storage. We could have a remote disk server that provides each client PC with a small 'virtual disk' and allows the client to read and write sectors on it. Since all communications between the client and the server would be in terms of absolute sectors, it is up to the client to decide how to use its virtual disk. For instance, the client could build a normal MS-DOS file system on it or use it as a raw data disk (with absolute disk read and write calls).

This type of server would be simple to implement and would make it easy to isolate and protect the data belonging to each client PC. However, this also means that sharing data between clients becomes difficult. Even if a client is allowed access to another client's disk, it would be unable to interpret the contents properly since it is the other client and not the server that keeps track of what is stored on the disk, where and how. The clients could come to an understanding to resolve this problem, but then each client would get unrestricted access to the other's disk, which is also undesirable. In short, it would be easy to have either total security or free sharing but not any convenient combination of them. Another problem is that since all the housekeeping and maintenance associated with the virtual disk is done by the client, it would lead to a lot of traffic on the

network.

A **file server**, (Fig. 1.1) on the other hand, works on a much higher level. The server locally maintains a complete file system and allows clients to access files in it (open, read, write, etc. All I/O in this case is in terms of characters or blocks of characters belonging to different files. To the client, the server appears to provide a remote file system, not a physical disk. Leaving all the housekeeping to the server reduces network traffic. Using the higher level notion of files makes security and sharing easier since the files can be considered to be owned or shared by different clients and the server can then screen individual requests to decide how to respond. The system can be made much more flexible and easier to use than a disk server, but at the price of increased software complexity.

2.1.2 Statelessness

After deciding to have a server that provides access to files, we come to the question of how exactly the client and the server are to interact. The simplest way would be to use the handle-based mechanism supported by DOS. i.e., The client contacts the server to open a file and obtain a handle. It then uses this handle to access the file and close it at the end. But this mechanism would work properly only over a communication channel that is 100% reliable. In practice, a packet sent over a network is always likely to get lost, duplicated, dropped or delivered out of order, especially if it has to cross several networks to reach its destination. The following examples show

the subtle effects of network unreliability on client-server interaction :

i. The server opens a file in response to a client's request and returns the handle, but the reply never reaches the client. In this case, the handle will never be used. Moreover the resources allocated for that handle will be wasted since the server cannot recover them by closing the file, without knowing whether the client has finished accessing the file or not.

ii. The client opens a file, receives a handle and tries to read, say, N bytes from the file. The server responds and updates the file pointer, but the reply gets lost. On timeout the client may send the same request again and the server will respond with the next N bytes. What the client receives is not what it asked for, but it has no way of knowing this.

These and similar problems arise because the server maintains local state information about requests ('open file', 'current position') and then interprets other requests on the basis of that information. Thus to work correctly, the server must be **stateless**, i.e., the response to a request must be completely independent of the previous history of requests. Note that this does not apply to the information **stored** in files (since the client expects to read back whatever it wrote), but only to the information **about previous requests.**

The requirement of statelessness affects the format of the request and reply messages. Obviously the request must contain a field indicating the operation to be performed. But since the client should not rely on the state of the server, each request must be self-contained. The client must not assume, and the

server must not maintain, any notion of 'current directory' or 'handle'. To be more specific, a request must always contain the complete pathname of the file being accessed and also the absolute position within the file, in case of I/O requests.

2.1.3 Statelessness vs. efficiency

The server has to depend on its local operating system to actually access the files requested by the client. If it is not allowed to maintain any state information at all, it would have to open the specified file, perform the required operation and close the file each time. But opening and closing files for each request would lead to a lot of overhead and make the server inefficient.

To have a stateless and yet efficient server, we must note that it is the interface between the client and the server that is required to be stateless. There is no reason why the server should not depend on the state of its local OS since, after all, they communicate through a highly reliable channel (the system call interface). The only requirement is that this state information must never be passed on to the client and the client must never assume that some state information is being maintained on its behalf. The server, while presenting a stateless face to the client, can maintain a set of open files internally for efficiency.

A good way of doing this would be to maintain a cache of recently accessed and opened files. Each entry in the cache should contain the full pathname of the file (for communication

with the client) and the file handle (for communication with the local OS). When a request arrives, the server looks in the cache first for the specified file. If there is a cache hit, it can do without having to open the file again. If not, the file is opened and an entry made in the cache.

Since there is a limit on the number of files that can be kept open by a process, the server will often have to choose a file from the cache to close before opening another one. The LRU (Least Recently Used) replacement policy is one that performs well on the average. Under this, the file that is selected to be replaced is the one that has been used least recently. An extra benefit of this is that if a client opens a file and then crashes, its file entry will quickly become the least recently used one in the server cache and get replaced at the very first opportunity.

2.1.4 Security and sharing

As mentioned before, since a file server carries out all its transactions in terms of files rather than disk sectors, incorporating features that ensure file security and allow proper file sharing is feasible. Files and/or directories could be associated with particular users who would be considered as their **owners**. The owner of a file should be able to access his files freely and decide if and how they are to be accessed by other users. Extending this further we could have **groups** of users, each of whose members could be allowed access to all or some of the other members' files. Users and groups could have predefined rights which determine what operations they are allowed to do in

their respective domains. These rights could be defined separately for different operations (read, write, delete, etc.) and over each file/directory. Thus being the owner of a file or a member of a group determines if that file can be accessed and the owner's/group's rights determine how it can be accessed.

When a file is permitted to be accessed by more than one user at a time, unexpected results may occur when two or more of them try to update or modify it simultaneously. To avoid these problems there must be some sort of file/record locking facility by which a user can request and be granted exclusive access to a file or a portion of it, during which period all other attempts to access it will fail.

Unauthorized users must be prevented from accessing the system. This is usually taken care of by a simple password mechanism at the time of logging in.

Finally there must be a special user, in charge of overall system management. In view of his responsibilities this user must have complete control over the whole system under all conditions. In particular, he must not be impeded by the restrictions of ownership/rights mentioned above.

2.1.5 Organization of the server

Having looked at the requirements of a file server, we can now see how they are implemented in the proposed system.

i. Basic structure

The server is implemented as a user process running under MS-DOS. It continually waits for a request from a client, performs the specified operation if possible and sends a reply. It maintains a complete MS-DOS file system, doing all file accesses through MS-DOS system calls. The server appears stateless to its clients, while internally maintaining a cache of recently accessed files for efficiency.

Access to the system is restricted to those clients who are registered as users. A user identifies himself at login-time by supplying a password. A special user called the superuser is responsible for the overall management of the system. Unless stated otherwise, none of the access restrictions to be discussed below apply to the superuser.

Each valid user has a directory of the same name in the root of the file system. He is considered the owner of this directory, its subdirectories and all files in them. Normally only the owner of a file is allowed to access it. But users can get together to form groups to facilitate file sharing.

ii. Rights

All file/directory operations are regulated by access rights that dictate which of the following file operations are allowed :

Open, Create, Read, Write, Delete, Search, Parental

(parental rights are required to create subdirectories).

These rights are defined separately for users, groups and directories. User rights and group rights determine what operations individual users and groups will be allowed to do and are set by the superuser. The owner of a directory decides which groups will be allowed to access files in it and sets the directory rights to indicate what operations they will be allowed to do. The owner can also declare any of his directories to be public, allowing all other users to access it (subject to the directory rights). Note that the directory rights do not place any restrictions on the owner's activities; he is controlled only by his user rights. Also, rights are not defined over individual files. This is done to reduce the amount of information the server has to keep track of.

Finally, at the lowest level, we have the file attribute security provided by MS-DOS in the form of the read-only attribute bit. This is a restriction even the superuser cannot override.

An example will help to clarify : if a user successfully logs in and tries to, say, delete a file from a directory he will be allowed to do so if -

He is the superuser
AND the file is not read-only

OR

He is the owner of the directory
AND he has the right to delete
AND the file is not read-only

OR

le has the right to delete

iND he is a member of a group with the right to delete

iND the owner has allowed this group to access his directory

iND the owner has allowed others to delete in his directory

iND the file is not read-only.

OR

le has the right to delete

iND the directory is public

iND the owner has allowed others to delete in his directory

iND the file is not read-only.

iii. Locking

To allow file/record Locking, requests to open a file in the shared mode and to lock/unlock records within a file (facilities provided by DOS) are supported. But this is one place where we have to compromise on our requirement of statelessness, since the server has to agree to keep a file open exclusively for a particular user. The danger here is that if the client crashes without closing the file, other clients may never be able to access it.

iv. Miscellaneous

Each user is assigned a quota of disk clusters with which he must meet his storage requirements. Although he can temporarily exceed his quota, he will not be allowed to login or logout while in that condition.

All MS-DOS system calls dealing with file and directory access are supported by the server. In addition, there are special calls to login and logout and to do user, group and directory inspection/maintenance. If any system call fails the relevant DOS errorcode is sent back to the client, allowing it to find out what went wrong.

All information pertaining to users, groups and directories is stored in separate binary files which are read into internal tables at startup. Modifications made to them are written through to the original files to ensure data consistency.

The information contained in the table entries is as follows:

User table entry -

- User name
- Password
- Rights
- Total clusters allocated
- Total clusters available

Group table entry -

- Group name
- Rights
- Members

Directory table entry -

- Directory name
- Owner
- 'Public' flag
- Rights
- Groups allowed access

2.2 The Redirector

In the proposed system, all client processes that access the server are supposed to be running under MS-DOS which, as such, has no facilities for communicating with the server. We have to develop suitable software to allow these processes to transparently access files on the server. By 'transparent access' we mean that all well-behaved programs (including the COMMAND.COM shell) running under MS-DOS and accessing local files through standard system calls should also be able to access remote files on the server without being aware of any difference. In this section we look at the redirector, the interface between user processes and the server, that makes this possible.

2.2.1 Transparent reference

The first question to be asked is how to refer to a remote file in a transparent manner. Under MS-DOS the complete path specification for a file fname in a directory dir on the drive "d" would be as follows :

d:\dir\fname

Obviously the remote file specification must be compatible with this naming scheme.

There are two choices : one is to have a special subdirectory called, say, \REMOTE which would logically have the entire remote file system as its child. Then a file \dir\fname on the server could be referred to as

d :\REMOTE\dir\fname

The other choice is to map an unused drive letter to a remote directory. Thus if, say, drive G is mapped to \dir on the

server, the above file specification would become

G:\fname

The second method has been chosen for the following reasons:

i. It is more general, in the sense that different drives can be mapped to different remote directories, possibly on different servers as well. This could also be done with the first method but then the pathnames tend to get too long and references to the file will be cumbersome. With the second method, by contrast, the path specification can actually be made shorter by mapping a single drive letter to a long remote path.

ii. This method will not look strange to DOS users since the SUBST command achieves quite the same thing with local files. (But there the only objective is to save a lot of typing.)

iii. This method is better from the point of view of efficiency too, since determining if a file is remote or not involves checking just one character (the drive letter) as against a string in the first method.

Note that a drive mapped in this manner must be treated as a "network drive" (a logical drive), not as a physical one. Programs like FORMAT and CHKDSK that try to do absolute disk accesses on the drive should, and will, fail.

2.2.2 Transparent access

Having decided how to refer to a remote file transparently we can consider the problem of actually accessing it.

For a remote disk server, the solution would be simple : install a special block mode device driver that responds to

MS-DOS requests for sector i/o by contacting the server. But since we have decided to have a server that provides file-oriented service, this solution will not be good enough for the very same reasons outlined in Section 2.1.1.

Instead we have chosen to intercept all file access requests made by the user process and, depending on the kind of request (local or remote), pass them on to the appropriate routines (DOS or the server) for handling.

Thus the following tasks have to be done by the redirector :

i. Intercept all file access requests and filter out those that have to do with remote files. Since all file access requests have to pass through the DOS system call interface (int 21h), this would be a good point to do the actual interception. Determining if the file is local or remote is done by simply checking the drive letter, as given in the previous section.

ii. Since the server is stateless, the redirector has to be responsible for **maintaining local state information** about the remote files being accessed by the calling process. This state information will be used when compiling a request to send to the server and must be updated on receiving the reply. For example, on opening a file the filename, access mode, file pointer position and handle must be recorded. For every subsequent i/o operation, the access mode must be checked to make sure that the requested operation is allowed and the file pointer position must be updated.

iii. **Translate** requests between the MS-DOS format and the self-contained stateless format supported by the server. The details of the translation will depend on the actual function.

-Requests involving file/directory names will need to have their names prefixed with the remote path represented by the specified drive. For this the drive-path mappings have to be stored in a **map table** which can be modified by appropriate utilities.

-For requests involving handles, the redirector has to store the full pathname of the file locally when the file is successfully opened, and retrieve it for all further accesses.

-For i/o requests the redirector must keep track of the file pointer position.

2.2.3 Implementation

In terms of actual implementation the redirector is a memory-resident routine through which all system calls are routed. Only local requests are allowed to pass through to DOS; remote requests are handled by interaction with the server and invalid requests are returned signaling an error. In addition, it provides an interface for the special calls for user, group and directory inspection/maintenance, that are redirected to the server. These calls will generally be issued by the utilities (discussed later).

The boundary between local drives and those that can be mapped to remote directories is decided by the number of logical drives in the system (set by the 'LASTDRIVE = ' line in CONFIG.SYS). Drives below this are assumed to be local drives; any of the remaining can be used for mapping.

2.3 Utilities

The system consisting of just the file server and the redirector would not be of much use without some utilities to assist in setting up and configuring. In addition to the basic utilities to login, logout or change network drive mappings, we need ways of conveniently inspecting and modifying the information about users, groups, directories and their rights, etc. In view of the large number of such utilities and their varied arguments and parameters, it would be very convenient to combine them all into a single menu-driven user-friendly package that can be used with the minimal amount of typing. The utilities for the system were developed with this in mind.

The utilities currently supported are -

i. Drive mapping :

- View current mappings
- Create a new mapping
- Cancel a mapping

ii. Inspect/modify :

- Users :
  - User rights
  - Group membership (inspect only)
  - Cluster allocation
  - Password (modify only)
- Groups :
  - Group rights
  - Members

Directories :

Owner (inspect only)

Directory rights

Restrict access/make public

Groups allowed access

iii. Log :

Log in

Log out

All these utilities involve special calls to the redirector which passes them on to the server.

2.4 The Network driver

In the previous sections we have been discussing the interaction between client and server, two distinct processes running on separate machines, without saying how exactly they achieve this. The network driver plays an important part in allowing these processes to communicate, by transporting data in the form of messages between them across the network.

2.4.1 The need for a driver

Basically the driver forms an interface between processes and the PC-LAN card, allowing them to send/receive messages to/from one another without having to know anything about the underlying network architecture. This makes these processes **independent of the actual protocols and hardware used for communication.** Any change in hardware or protocols will affect only the driver itself and not the processes, so long as the interface between them and the driver stays the same.

A process that wishes to send a message to another process should only be expected to write it to the driver, specifying the destination. The driver copies it into an internal buffer and assumes all responsibility for ensuring that it reaches its destination. Similarly incoming messages are verified for integrity and buffered internally. When a process reads the driver,it should get a copy of the first message that arrived since the last read, along with information about where it came from.

2.4.2 Protocol adopted

Things are easy if the client and the server are on a single network. The data could simply be encapsulated in a PC-LAN packet and written to the PC-LAN card. However, to be more useful, the client and the server should be able to communicate across different interconnected networks. Thus we have to use some sort of internetworking protocol above the PC-LAN physical transport mechanism.

Since most projects developed around the PC-LAN have used the DARPA Internet Protocol (IP) [3] [4], that is the one we will use. For flexibility, we have decided to send messages as datagrams using the User Datagram Protocol (UDP) of the DARPA protocol suite. This protocol can be used to distinguish between multiple recipients on a single machine. Each machine would be assumed to have a set of abstract destination points called ports at which incoming messages are queued until a process reads them. Even if the network hardware provides reliable delivery, internetworks do not guarantee reliable message transport. Packets can be lost, duplicated or delivered out of order when traveling across networks. UDP provides unreliable delivery. So the upper levels of software must use acknowledgments and retransmissions to ensure that messages arrive properly.

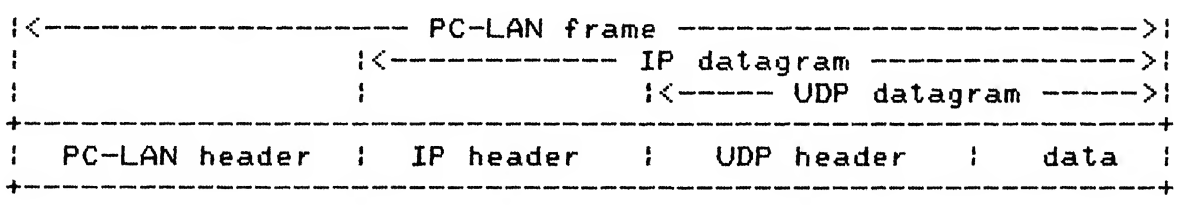
This is not much of a problem from the point of view of client - server interaction since the server always replies to a a request, even if no data is to be sent. Thus when a client sends a request, the reply itself forms the acknowledgment. If the reply does not arrive before the timeout expires, the client will retry the request by sending it again. This does not do any

harm since the server is stateless. Thus acknowledgments and retries are an inherent part of client - server interaction and we do not really need to use any special protocol (like TCP) that ensures reliable transmission by acknowledging and retrying individual messages.

2.4.3 Implementation

The network driver is implemented in the form of an installable device driver which the process opens and reads/writes to receive/send messages.

Providing the physical transport mechanism is the PC-LAN card. (See the appendix for information about using the card). Since messages are to be sent as UDP datagrams, the data field of the PC-LAN packet will contain an IP datagram which in turn contains a UDP datagram in its data field. The data field of the UDP datagram will contain the actual data.



The standard device i/o calls pass only a pointer to the data and the number of bytes to be transferred. But we need to pass more information - like the destination internet address, the destination port number and the source port number. To allow for this our device i/o calls expect a pointer to a control block which contains all the necessary information, including a

pointer to the data itself. The structure of the control block is as follows :

foreign host internet address

foreign port number

local port number

ack flag

pointer to the message to be sent

pointer to a buffer to hold incoming messages

The ack flag is used to indicate whether an acknowledgment is expected in response to the message being sent. If this flag is not set a write call returns immediately after the message has been sent. Otherwise the driver waits for a message to arrive at the specified local port. On timeout the user is given the option of retrying, in which case the message will be sent again. Obviously this facility will be used only by the clients; the server does not need it.

The driver has a pool of buffers into which incoming packets are read in. Once a packet is read in, it is verified and then enqueued at the port specified in its UDP header. Since reception of packets from the PC-LAN card is interrupt-driven, all this is done in the background.

Opening and closing of local ports is done through ioctl calls.

A process opens the driver and then opens a port on it. To send a message it issues a write call, passing a pointer to a control block which specifies the foreign host address, the foreign port and the local port to which replies must be

addressed. For a read call the driver dequeues the first message waiting at the local port specified in the control block and copies it out to the specified buffer. The driver also extracts the source address and source port information from the header of the message and copies them into the foreign host address and foreign port fields of the control block, so that the calling process can know where the message came from.

CHAPTER 3

IMPLEMENTATION DETAILS

3.1 The Server

The server is implemented as a user process running under MS-DOS and interfacing with the network through the driver. It handles incoming requests by translating them to DOS system calls and returns the relevant information.

The request message structure is defined in "freq.h" :

```
typedef struct
{     unsigned seq_no;            /* sequence no. of req.*/
      unsigned id;                /* user identification */
      char name [MAXNAMLEN];      /* file/directory name */
      int code;                   /* operation code/error*/
      long pos;                   /* offset within file  */
      unsigned len;               /* length of data field*/
      unsigned extra;             /* extra parameter */
      char data [FMAXDAT];        /* data */
} FREQ;
```

The operation codes supported are :

OPEN, CLOSE, READ, WRITE, - file i/o

DELETE, RENAME,

CHDIR, MKDIR, RMDIR,

SEARCHF, SEARCHN, - search for matching files

CREAT, CREATNEW, CREATTEMP, - file creation

CHMOD, GETMOD, - file attributes

SETDATE, GETDATE, - file date and time

FLEN, GETSPACE, - file length, disk space

LOCK, UNLOCK, - record locking/unlocking

UTILS - various utilities like :

LOGIN, LOGOUT,

GETUSRINFO, GETGRPINFO, GETDIRINFO,

SETUSRINFO, SETGRPINFO, SETDIRINFO,

GETUSRLIST, GETGRPLIST, GETUGRPS,

MKUSER, MKGROUP,

RMUSER, RMGROUP

3.1.1 The server main program (fs.c)

The main program consists of an initialization stage followed by the main loop in which control stays until shutdown. The initialization includes :

i. initializing the driver.

ii. allocating memory for all the necessary tables and reading in the user, group and directory information from files on disk.

iii. allocating memory for the file cache and initializing it.

If any of these fails, the program aborts after printing the relevant error message on the console.

In the main loop the server repeatedly waits for a request (**getreq**), processes it and sends the appropriate reply (**reply**). The processing consists of the following steps :

i. The operation code specified in the request is checked for validity.

ii. The function **access_chk** is invoked to ensure that the user making the request is allowed to perform the operation in the specified directory.

iii. If any of these preliminary checks fail, the code field is set to the relevant error code. Otherwise, the handler routine corresponding to the operation requested is called and the code field is set to the code returned by it : 0 for success and the relevant DOS error code for failure.

3.1.2 The driver interface (fsio.c)

The **dev_init** function, which is called at the initialization stage, is responsible for opening the device driver and opening a port on it for use by the server. It also initializes the control block and registers the function **close_dev** to be called when the program terminates normally. **close_dev** merely shuts down the port and the device.

The routines **getreq** and **reply** form the device interface. **getreq** simply reads the device and returns a pointer to any incoming message, or NULL if none are waiting at the port. **reply** receives a pointer to a reply message to be sent to the client and writes it to the device. These routines do not have to keep track of the address of the client; this is automatically taken care of by the way the control block is defined. (Sec. 3.3.6)

3.1.3 Access control (fac.c)

This file contains all the code and data structures that control access to the server file system.

i. Tables

Information about users, groups and directories is maintained in the tables **utab**, **gtab** and **dtab** respectively. The structure of each of their entries is reproduced below :

```
typedef struct user
{     char name [MAXUNAM];      /* user name */
      char pwd [MAXUNAM];       /* password  */
      unsigned rights;          /* user rights */
      unsigned free;             /* no. of free clusters */
      unsigned tncl;             /* total clusters allocated */
      unsigned scn;             /* starting cluster no. of
                                   user's directory */
} USER;
```

```
typedef struct group
{     char name [MAXUNAM];      /* group name */
      unsigned rights;          /* group rights */
      unsigned mems [MAXGRM];   /* members */
} GROUP;

typedef struct dir
{     char name [MAXNAMLEN];    /* directory name */
      unsigned owner;           /* owner of this dir. */
      unsigned ispublic;        /* 'public' flag */
      unsigned rights;          /* dir. rights */
      unsigned groups [MAXUGR];/*groups allowed access*/
      struct dir *next;         /* pointer to next entry */
} DIR;
```

Since users and groups are identified by their numbers, the user and group tables are organized as linear arrays and the correct entry is located by simple indexing. But for the directory table, a different approach must be used since a directory is identified by its name and a linear search of the entire table would be prohibitive.

The solution chosen is to organize it as a hashed table. Thus the directory table is a linear array of pointers, each of which points to a linked list of directory structures. To locate a directory the hashing function is applied to its name to get an integer which is used to index into the directory table. If the directory exists it will be found in the linked list there, which can be searched quickly. The hash function used here (**hash**) is quite simple and reasonably good - it just takes the sum of the characters in the directory name modulo the directory table size (which is a prime number). The whole lookup operation is done by the **getdir** function which accepts a directory name and returns a pointer to its entry if it exists or NULL if it doesn't.

ii. Logging in and out

This module also contains the routines that allow users to log in and log out. On logging in, a user gives his user no. and password. After ensuring that these are valid the server makes an entry in its **logtab** array for future identification and returns a unique number (actually a pointer to the **logtab** entry) to the user with which he identifies himself in all further contacts. When the user logs out this entry is invalidated.

iii. Access checking

An important routine in this module is **access_chk** which has the responsibility of rejecting illegal requests made by users. It does this using the information in the user, group and directory information tables and an array **rtab** that specifies the rights needed for each operation. The **access_chk** function first identifies the user making the request by finding his user no. through the **logtab** entry. It then checks if :

i. the user has the right to perform the requested operation.

ii. the directory in which the operation is to be performed exists. This is done by looking it up in the directory table using the **getdir** function.

iii. the user is allowed to do the operation in this directory. This will be true if :

- the user is the superuser

OR

- the user is the owner of the directory

OR

- users other than the owner are allowed to do this

operation in the directory

AND the directory is public

OR

- users other than the owner are allowed to do this operation in the directory

AND the user is a member of a group with the appropriate rights

AND this group is allowed access to the directory

If, after these checks, the request is found to be legal the function returns a value of 0 to indicate success. Otherwise it returns the error code corresponding to 'access denied'.

iv. Table maintenance

The functions **utab_io, gtab_io** and **dtab_io** take care of all i/o between the user, group and directory tables and their corresponding files on disk. When the server is initialized these functions are called with the INIT option, allowing them to load the information into memory. When an entry is modified, the corresponding function is called with the UPDATE option to enable the change to be written through to the original file.

When a user creates/removes a subdirectory, the changes must be entered in the directory table to ensure correct response to future requests for it. This is done by the **facmkdir** and **facrmdir** functions which install/remove entries in the directory table. **facmkdir** assumes that the newly created directory belongs to the parent directory's owner and sets the fields of the entry so that all other users are denied access to it. There is also a

collection of functions that allow users to inspect or modify their user, group or directory entries. **getusrinfo, getgrpinfo and getdirinfo** return the requested entry from the respective table while **setusrinfo, setgrpinfo and setdirinfo** replace the entry with the supplied one. An ordinary user can set only those fields over which he has control. These are :

- for the user table : the password
- for the group table : none
- for the dir. table : rights, the 'public' flag and groups allowed access

getusrlist and **getgrplist** return a list of names of all valid users/groups in serial order. **getugrps** returns the numbers of the groups to which the user belongs.

The next group of functions allow the superuser to install or remove users and groups. **mkuser** creates a new user table entry with the name specified in the name field of the request. It also creates a subdirectory of the same name in the root of the file system. The password and rights of the new user must be set separately by the superuser with **setusrinfo. rmuser** removes a user from the system by deleting his entry in the user table. It does not remove any files or directories belonging to the user.

Similarly **mkgroup** and **rmgroup** are for creating and removing groups respectively. The rights and members of a newly formed group must be set separately by the superuser using **setgrpinfo.**

3.1.4 The file cache (fch.c)

i. Organization

The cache is organized as a doubly linked circular list of entries. Each entry is of the form :

```
typedef struct fileinfo
{     char name [MAXNAMLEN];     /* the filename */
      int hdl;                   /* handle */
      int mode;                  /* access mode */
      int uno;                   /* user who opened it */
      int lock;                  /*whether locked or not*/
      long pos;                  /* current offset */
      struct fileinfo *next;     /* pointers to other
      struct fileinfo *prev;        entries */
} C_FILE;
```

The 'lock' field indicates that the corresponding file has been opened in a shared mode and therefore must not be closed until requested to by the user who opened it.

One of the entries in the cache is never used but forms the HEAD of the cache. The cache algorithm is such that HEAD->next is always the entry of the least recently used file (the OLDEST) while HEAD->prev is always that of the most recently used one (the NEWEST). Thus if we start at the NEWEST and follow the 'prev' links we can run through all the entries in the order of their recent access and finally reach the OLDEST.

ii. File access routines

The file access routines provided by this module allow the server to access files without knowing about the cache; i.e. the cache is transparent to the server so long as it uses the following primitives to access files :

c_open, c_seek, c_read, c_write, c_close

c_open is used to open a file - it returns a pointer to the cache entry (an object of type C_FILE), which is used to access the file. It searches the cache to see if the user has already opened the file in the specified mode. If so, it makes this cache entry the NEWEST by calling **promote** and returns a pointer to it. If not, an attempt is made to open it through a system call. If this fails (because the file doesn't exist or because access has been temporarily denied due to a sharing violation), **c_open** returns NULL. Otherwise **getfree** is called to get a free slot in the cache into which all the state information of the file is stored. This entry is then made the NEWEST. By this process, a file that is accessed very often always stays in the cache as one of the newest and a file that is not accessed for a long time quickly becomes the OLDEST and thereby, the prime candidate for replacement.

c_seek sets the file pointer position. If the specified position is the same as the current one (as recorded in the cache entry), it returns immediately. Otherwise the position is set through a system call and the cache entry updated.

c_read and **c_write** transfer the requested number of bytes and update the file pointer position in the cache entry.

iii. Cache maintenance routines

getfree returns a pointer to a free slot in the cache. It searches the cache starting from the OLDEST until it finds an entry that can be used (i.e. one that is not locked). If the file is open it is closed and a pointer to this entry is returned.

that can be safely closed.

promote is called when a file is accessed to give it a new lease of life in the cache. It simply removes the entry from its position in the list and inserts it between the HEAD and the NEWEST, thus making it the new NEWEST. Similarly **demote** makes a file the OLDEST by inserting its entry between the HEAD and the current OLDEST. This file then becomes the least recently used one.

uncache is used to purge a file from the cache when its name is known. It locates its entry in the cache and calls **c_close**, which closes the file, invalidates the entry and explicitly makes it the OLDEST by calling **demote**. **c_init** is called to initialize the cache by allocating memory for it and setting up the doubly linked circular list structure.

3.1.5 The request handlers (frh.c)

This module contains the handler routines that actually perform the operations requested by users after the server has checked their permissions and rights.

i. A typical handler

A typical handler receives a pointer to the request message. It performs its operation using the necessary parameters from the request header and returns a code to indicate success or failure. For success the return code is 0, while for failure it is the DOS errorcode. The handler is responsible for setting the len field of the header if the reply is to contain any data.

ii. OPEN, CLOSE, READ, WRITE

As seen in the previous section the server does all file accesses through calls to c_open, c_seek, c_read and c_write. The READ and WRITE requests are handled by fsread and fswrite respectively. They both call c_open to make sure the file is open and c_seek to make sure the file pointer is positioned correctly before calling c_read/c_write. Before returning they set the extra field in the reply header to the no. of bytes transferred and update the pos field. fsread has to send data along with the reply, so it also sets the len field to the no. of bytes; fswrite sets it to 0.

iii. DELETE, RENAME

The DELETE and RENAME requests support the wildcard character '?' in the filename. fsdelete first tries to delete the file straightaway. If it succeeds, or if it fails and the errorcode is 'access denied', it can return immediately. Otherwise if the filename contains the character '?' it tries to delete the file(s) with the fcb delete call (int 21h, fn.13h). fsrename follows the same logic.

iv.CHDIR, MKDIR, RMDIR

fschdir simply returns OK since the server does not maintain any 'current directory'. The only point in routing this request to the server is to make sure that the directory exists and the user is permitted to change to it. This has already been done by the access_chk function, so there is nothing left for fschdir to do.

fsmkdir and **fsrmdir** perform their obvious functions and also update the directory table by calling **facmkdir** and **facrmdir** respectively. (Sec 3.1.3).

v. SEARCHF, SEARCHN

fssearchf and **fssearchn** are the counterparts of the DOS functions 4eh (search for first match) and 4fh (search for next match). The difference is that they support buffering of search information, for the sake of efficiency. i.e. These functions return search information about as many files as possible.

fssearchf expects the full pathname of the file while fssearchn needs the search information from the previous search, since the server is stateless. The number of matching files whose information is being sent in the reply is placed in the extra field of the reply header.

vi. CREAT, CREATNEW, CREATTEMP

fscreat, **fscreattemp** and **fscreatnew** are the counterparts of DOS functions 3ch, 5ah and 5bh respectively. **fscreattemp** returns the name of the newly created file in the name field of the reply.

vii. Miscellaneous

The server also supports requests to get and set the attributes of a file (GETMOD, CHMOD) and its date and time (GETDATE, SETDATE).

The request FLEN is used by the redirector to find the length of a file. **fslock** handles the LOCK and UNLOCK requests for record locking.

3.2 The redirector

The redirector is a memory resident routine that provides user processes with transparent access to remote files on the server. It achieves this by chaining itself into the int 21h vector (the entry point for all system calls) and redirecting all requests for remote files to the server. It takes care of local state maintenance and translation between DOS and server formats. In addition it provides an interface for direct communication with the server through the DOS multiplex interrupt (int 2fh). This is mainly for the utilities intended to be used with the system, including for logging in and logging out. Some of the utilities (GETMAP, SETMAP, DELMAP, GETUNO) are handled by the redirector itself.

3.2.1 The main program (rdr.asm)

The main program is concerned with installing the resident portion of the redirector in memory and initializing the multiplex interrupt handler. First it makes sure that it has not been installed already (by calling **chk_install**) and that the network driver is accessible (through a call to **chk_dev**). It the chains itself into the multiplex interrupt vector (**install**) and calls the DOS function 31h to terminate and stay resident. At this point the utilities can be used to try to login to the server. The redirector proper will not be initialized until this is done.

3.2.2 The multiplex interrupt handler (rmux.asm)

The multiplex interrupt handler used by the redirector (**rmux**) uses the multiplex number 80h. On getting a valid call

(i.e. with AH = 0) it pushes all the registers onto the stack and calls the function **rutils**, passing it a pointer to the registers on the stack. **rutils** is responsible for handling all the utility calls. However the call with AL = 0 (which must be supported by all handlers) returns immediately with AL = 0ffh.

The **chk_install** function is called by the main program to see if the redirector (more specifically, **rmux**) has already been installed. It simply issues an int 2fh with AH = 80h and AL = 0. The return value in AL will be 0ffh if the redirector is resident.

The **install** function installs **rmux** by saving the original int 2fh vector and then setting it to point to **rmux**.

3.2.3 The redirector interrupt handler (rint.asm)

rint is the routine that actually intercepts system calls and redirects the ones having to do with remote requests to the server. The int 21h vector is set to point to it when the user logs in. The original int 21h vector is saved as the vector for int 62h (which is normally unused), allowing the redirector itself to call DOS through an int 62h.

rint first saves all the registers on the stack and calls **whereto**, passing it a pointer to the saved registers. The value returned by **whereto** indicates whether :

i. the call is purely local, in which case control is turned over to DOS (through an int 62h) after popping all the registers off the stack.

ii. the call is to be redirected to the server, which is done by

calling **remotecall**. When **remotecall** returns, the registers on the stack contain the reply to the call in MS-DOS format. All that **rint** needs to do is make the carry flag reflect the status (success/failure) of the call, pop off all the registers and return to the calling process.

iii. the call is in error and cannot be processed. This can happen if, say, the drive specified is neither a local one nor a drive mapped to a remote directory. In this case **rint** pops off the registers, sets the carry flag and returns to the calling process.

3.2.4 Where to send a request (rwhere.c)

Since DOS supports the notion of 'current' drive, the redirector must keep track of it to properly handle calls that do not specify the drive. The variables **n_drvs** and **cur_drv** contain the number of logical drives in the system and the current drive respectively and are initialized at login.

whereto is the function that analyses a system call to decide where it should be sent to - DOS, the server or back to the calling process with an error code. For calls with an ASCIIZ path specification this is done by checking the drive (or the current drive if none is specified). If the drive is local (<= **n_drvs**), the call must be passed on to DOS; if it is a previously mapped remote drive, it goes to the server. Any other drive is invalid. Drive mappings are maintained in a map table (Sec. 3.2.6).

For calls involving file handles, **whereto** uses the handle to index into the **fptab** table, which for remote handles will contain

a pointer to a structure containing the file state information. For unallocated or local handles the entry will be NULL. (Sec. 3.2.7 (i.))

The redirector does not, in general, redirect FCB-oriented calls. However the FCB calls 11h and 12h (search), 13h (delete) and 17h (rename) have to be handled for the sake of transparency because the DOS shell COMMAND.COM uses them.

In addition to the above calls, **whereto** also intercepts function 0eh (set drive) to keep track of changes in the default drive, and function 19h (get drive) to return the default drive.

remotecall is the interface between the interceptor **rint** and the various handler routines (Sec. 3.2.7) that see to each remote function call. The handlers are invoked through a call table (**rcall_hndlr**) indexed by the function number (in AH). **remotecall** returns 0 if the call succeeds. If it fails it sets the saved AX register on the stack to the errorcode returned by the handler and returns 1.

3.2.5 The device interface (rio.c)

netio forms the interface with the network driver. It writes the message it is passed to the driver. The ack field in the device control block is set, so this driver call does not return until a reply is obtained from the server. **netio** returns a pointer to this reply.

servercall is the function used by the handlers to send requests to the server and obtain replies. It accepts a request message from a handler, fills in the user's id and sends it to the server by calling **netio**. If the call succeeds it returns a

pointer to the reply message, otherwise it sets the global variable **errcode** and returns NULL.

3.2.6 Drive mappings (rmap.c)

This module contains the **map** table and functions that use or modify it.

The **map** table is a 26-element array of 64-byte strings (one for each drive), each string containing the remote path to which its corresponding drive is mapped. Unmapped drives (and local ones) have NULL strings in the **map** table.

rmap is the routine that manages the **map** table through the GETMAP, SETMAP and DELMAP calls issued by the utilities. To SETMAP, it first makes sure that the path exists by requesting the server to CHDIR to it and noting the response. It then copies the remote pathname into the **map** table. GETMAP simply copies out the appropriate **map** table entry to the user's buffer while DELMAP cancels a mapping by writing a NULL into the **map** table entry.

The **pathmap** function takes the path specified by the caller and builds the complete absolute path specification to be sent to the server by prefixing it with entries corresponding to the specified (or default) drive from the **map** table and the current directory table **cur_dir** (if the specified path is not absolute). It uses **pathcpy** to take care of any '.' or '..' fields in the specified path, since these are not recognized by the server. **pathmap** returns a pointer to the complete path, or NULL in case of an error in the path.

For example, <drv>: \ <path> \ <filename> gets expanded to

\ <mapped directory> \ <path> \ <filename>

while <drv>: <path> \ <filename> becomes

\ <mapped dir.> \ <current dir.> \ <path> \ <filename>

fcbmap provides the same service for files specified through FCBs.

3.2.7 The request handlers

A typical handler receives a pointer to the registers saved on the stack (containing the call parameters in DOS format). It compiles a request message with the proper parameters, using **pathmap** to expand any path specification, and sends the call through **servercall**. The reply is then translated back into DOS format and returned to the caller through the registers on the stack. It returns the code returned by **servercall**.

i. File requests (rfrq.c)

ropen, the handler for DOS function 3eh, opens a remote file by sending an OPEN request to the server specifying the complete path and the mode. On success it calls **lopen** to allocate a file handle which it returns to the caller.

lopen gets a file handle from DOS through **halloc** and also enters the file state information (name, mode, position) into an empty L_FILE structure allocated by **falloc** from the file state table **ftab**. The pointer to this structure is saved in the file pointer table **fptab** with the allocated handle as index. Thus for future accesses, **fptab**[<handle>] will point to the L_FILE structure of the file from which all the necessary state

information can be obtained. The structure L_FILE is defined as follows :

```
typedef struct
{     char name[MAXNAMLEN];     /*   complete path spec   */
      int mode;                 /*     access mode        */
      int nhndls;               /*   number of handles    */
      long pos;                 /* current offset in file */
} L_FILE;
```

rread (function 3fh) and **rwrite** (function 40h) may have to call the server several times to transfer data since the amount of data that can be transferred through a single call is limited. First, they locate the L_FILE structure of the file, check the access mode to make sure that the transfer requested is permitted and get the current file pointer position. Then they repeatedly call the server with READ/WRITE requests respectively, updating the buffer pointer and the byte count between calls, until the specified number of bytes have been transferred. The file pointer position gets updated automatically since the server always returns the new position in the reply.

rdel(functions 13h, 41h) and **rrename**(functions 17h, 56h) do their jobs by sending DELETE and RENAME requests respectively.

rcreat (function 3ch), **rcreattemp** (function 5ah) and **rcreatnew** (function 5bh) translate to CREAT, CREATTEMP and CREATNEW requests respectively.

Strictly speaking, **rseek** (function 42h) does not have to contact the server since the file pointer position is maintained, and can be updated, locally. However when the position specified is with respect to the end of the file, the current length of the file must be obtained through an FLEN request.

ii. Directory requests (rdrq.c)

The **cur_dir** table in this module contains the current directory corresponding to each drive.

Function 47h (get current directory) is handled locally by **rgetdir**, which copies out the appropriate entry from the **cur_dir** table into the caller's buffer.

rchdir (function 3bh) first checks if the directory exists and the user is permitted to change to it by sending a CHDIR request to the server. On success, the portion of the path after the map field is copied into the **cur_dir** table.

Functions 39h and 3ah are handled by **rmkdir** and **rrmdir** respectively through MKDIR/RMDIR requests.

iii. Search requests (rsrq.c)

rsearch handles the functions 4eh (search for first match) and 4fh (search for next match) through SEARCHF and SEARCHN requests respectively. SEARCHN must be accompanied by the search information from the previous search request since the server is stateless.

For efficiency, these calls are buffered, i.e. the reply to a SEARCHF/SEARCHN request will contain search information about more than one matching file (if there are any). The limit is set by the maximum amount of data a reply message can hold (FMAXDAT = 143 currently). Since the DOS search information is a 43-byte structure, this means that information about 3 files can be contained in a reply. Thus the redirector has to contact the server only on every third search call it receives.

For a function 4eh call, the complete path is sent in a SEARCHF request. The extra field of the reply header contains the number of matching files returned. **rsearch** copies the first of them into the current DTA and the remaining into its local buffer **sbuf**. The next 4fh calls are satisfied by copying out the information from **sbuf** until it becomes empty. When this happens a SEARCHN request is sent and the buffer filled again.

rfcbsearch works on the same lines for the FCB search calls 11h and 12h. The main difference is that the return information expected by the caller is in a different format. Instead of directly copying out the information **rfcbsearch** calls **xlate** to parse it into the fields of an unopened FCB in the DTA.

iv. Miscellaneous requests (rmrq.c)

Handlers for the utilities are defined in this module.

rutils, called by the multiplex interrupt handler, invokes the appropriate handler through a call table **util_hndlr**. The user/group/directory information/maintenance calls are handled by **rinfo** which passes them onto the server and returns the reply. For these calls, the user/group no. is expected in CX and DS:DX points to the buffer. The MAP calls are handled locally by **rmap** (Sec 3.2.6).

The LOGIN call is handled by **login**. It expects the server's internet address in SI:DI, the user no. in CX and a pointer to the password in DS:DX. After initializing the driver (**dev_init**), it attempts to login through a call to **rlogin**. On success it sets the **log** flag and initializes the **n_drvs** and **cur_drv** variables. It then activates the redirector by pointing the int 21h vector to

the redirector interrupt handler **rint**. The original int 21h vector is saved as the vector for int 62h.

rlogin saves the user number in **uno** and sends a LOGIN request to the server. On success the user id returned by the server is saved in **uid** for future communications.

logoff sends a LOGOFF request through **rlogoff** and 'switches off' the redirector by restoring the int 21 vector. It also clears the **log** flag and shuts down the driver.

3.3 The driver

Before looking at the actual driver some of its support routines and data structures must be explained.

3.3.1 Buffer management (buf.c)

This module maintains a pool of buffers into which incoming packets are read. **allocp** returns a pointer to a free buffer from the pool after marking it RESERVED, or NULL if no free buffers are available. When the buffer is no longer needed **freep** must be called to release it by marking it FREE.

3.3.2 Queue management (q.c)

A queue is simply a linked list of buffers, with pointers to the head and the tail. **enque** adds a packet to the tail of a queue while **deque** removes the packet at the head and returns a pointer to it, or NULL if the queue is empty.

3.3.3 Port management (port.c)

Incoming packets found to be OK are queued at the port specified in its UDP header. A port consists of a buffer queue and a flag indicating the port status (FREE or RESERVED).

A port is opened by a call to **popen**, specifying the required port number (in the range 1 - NPORTS) or -1 to get the first available port. **popen** returns -1 if the allocation cannot be made. **pclose** clears all waiting packets at the specified port and closes it.

A packet is sent to a port by a call to **psend**, which **enques** it. **precv** **deques** a packet from its queue and returns a pointer to

it. If there are no packets it retries NTRIES times before giving up and returning NULL.

3.3.4 The PC-LAN interface (pclan.asm)

This module contains the routines used by the driver to interface with the PC-LAN IMP card, details of which can be found in the appendix.

netout accepts a pointer to a packet with the dest, type and len fields properly filled in and writes it to the IMP card. It returns 0 for success, or 1 if the transmit buffers were full and the packet could not be sent.

Reception of packets is handled in the background by the IRQ2 interrupt routine **netin**. On receiving the first byte of a new packet it calls **allocp** to get a free buffer, into which it reads each byte of the packet. The complete packet is forwarded to **demux**. If the packet is rejected for some reason **demux** returns 1, in which case its buffer is freed by a call to **freep**.

3.3.5 udp.c, ip.c, net.c

A UDP datagram is sent by copying the data into a packet buffer and passing it to **udpsend** along with its length and a pointer to a UDPCONN structure containing the foreign internet address, the foreign port no. and the local port no. **udpsend** fills in the UDP header of the packet, calculates its checksum and forwards it to **ipsend**, which in turn fills in its IP header and calculates its checksum and passes it to **send**. **send** sets the dest and len fields of the PC-LAN header appropriately and sends

the packet through **netout**.

When a packet is received it is passed on to **demux**, which checks the len and type fields of the PC-LAN header. Packets of non-zero length and type IP are forwarded to **ipdemux** and others are rejected. **ipdemux** checks to see if the received length of the packet matches the length as specified in its IP header and verifies the checksum of the IP header. UDP datagrams are passed on to **udpdemux**, which verifies its length and checksum and sends it to the port specified in its UDP header by calling **psend**.

3.3.6 The device driver (netdrvr.asm)

The driver is an installable character mode device driver (NET). It needs the OPEN/CLOSE and IOCTL calls and so must be used with MS-DOS version 3.00 or higher. All i/o is done by passing a pointer to a control block :

```
cblk struc
     faddr     dd   ?        ; foreign internet address
     fport     dw   ?        ; foreign port number
     lport     dw   ?        ; local port number
     ack       dw   ?        ; acknowledge flag
     xbuf      dd   ?        ; pointer to message to be sent
     rbuf      dd   ?        ; pointer to buffer to
                             ; hold incoming message
cblk ends
```

Note that since the control block uses foreign and local specifications instead of destination and source, a process can send a reply to a message without having to exchange any source and destination fields.

i. OPEN/CLOSE calls

The **open** and **close** routines ensure that as long as the driver is open, the IRQ2 vector points to the packet receive

routine **netin** and the IRQ2 mask bit in the interrupt mask register of the 8259A is clear, thus enabling packet reception.

ii. IOCTL calls

IOCTL calls with a pointer to the control block are used to open and close ports on the device. A read call opens the port specified in the lport field. lport is set to -1 if the port could not be opened. If any port is acceptable lport should be set to -1 before making the call. In that case lport contains the number of the opened port on return. A write call closes the specified port.

iii. READ/WRITE calls

read calls **precv** with the local port specified in the control block. If it receives a pointer to a waiting packet, it extracts the length of the data, the foreign internet address and the foreign port number out of it. The foreign address and port no. are transferred to the control block while the data itself is copied out to the specified buffer. The buffer holding the packet is then released by calling **freep**. On the other hand, if the call to **precv** times out, **read** returns 0.

write first checks the length of the message to make sure that it is not greater than UMAXDAT. The data is then copied into an internal packet buffer and sent by calling **udpsend**, specifying the foreign address, foreign port and local port. If the ack flag is set, the local port is cleared and **read** called to receive the reply.

If this call to **read** times out, a NOT READY error is returned to DOS, which invokes the critical error handler, resulting in the familiar 'Abort, Retry, Ignore?' prompt.

3.4 The utilities

All the utility calls are menu-driven to make life easier for the programmer (since this approach makes it simpler to make modifications and eliminates the need for error checking on the input) as well as the user (since it saves a lot of typing and he does not have to memorize separate commands and parameters for each utility).

All the work involved in setting up and managing windows for each menu is done by the function **win** which accepts a pointer to a WINSPEC structure physically defining the window.

```
typedef struct
{     char *title;          /* title to be displayed */
      int rows;             /* no. of rows */
      int cols;             /* no. of columns */
      int type;             /* type of window */
      OPTIONS *o;           /* pointer to options */
} WINSPEC;
```

The o field points to an array of OPTIONS structures, which specify the options to be displayed in the menu and the functions that handle them.

```
typedef struct
{     char *text;           /* name of this option */
      void (*func) ();      /* handler for the option */
      void *par;            /* pointer to a parameter */
} OPTIONS;
```

win sets up the window as specified. For a MENU type window it displays the options and allows the user to run through them with the up and down cursor keys. When an option is selected with the 'enter' key its corresponding handler is invoked. This handler could simply do something and return, but it could also call **win** with its own WINSPEC and OPTIONS, thus creating a submenu. A window of type MESSAGE displays a message and waits while an INPUT window prints a prompt and waits for string input.

Pressing the 'esc' key exits a window.

getstring opens a window, displays its first argument string and reads input into its second argument string. **msg** displays a message in a window with a title. These functions actually translate to appropriate calls to **win** and are convenient for user interaction.

The above three functions are extensively used by the utilities. The end handlers for the options do their jobs by calls to the redirector through int 2fh, specifying the command and the relevant parameters. Only the superuser has access to all the options; an ordinary user is shown only those options that he is permitted to use.

CHAPTER 4
CONCLUSIONS

The system as described in this report has been successfully implemented with a PC-XT with a 20 MB hard disk used as the server and client software running on another PC connected to the PC-LAN. In addition to the COMMAND.COM shell and various DOS commands dealing with file access, several application programs and miscellaneous utilities were run on the client PC and were found to be working satisfactorily with no major difficulties in accessing the remote file system.

The overall file transfer rate is around 4 Kbps (measured for the DOS TYPE command). The low transfer rate is partly due to the software overhead incurred at the client and server ends, and partly due to the low maximum data rate (192 Kbps) of the present hardware and the small packet length (255 bytes) used currently. Owing to the limitation imposed by the packet length, even small files have to be broken up into several smaller sections to be transferred separately, thus significantly increasing the total overhead.

Scope for future work

i. In the present implementation, it is possible to have more than one server running independently on different machines. But the redirector does not allow a user to login to more than one server at a time. This restriction could be removed, providing for greater flexibility and better and more efficient resource sharing.

ii. The idea of a remote file system could be extended into a network file system, by allowing the server to honor requests for files which may not be on its own file system, but on any other PC on the network. The resulting system would provide complete transparency and convenience, with a user being able to access files anywhere on the network as easily as accessing his local disk. This could be done by the modification of the server to allow it to send requests itself (in addition to waiting for requests from clients and sending replies) and the introduction of a background routine on all client PCs, that gets activated by a request from the server for a file access. However this routine would have to take care not to re-enter DOS when making its system calls. This is one application where the provision on the driver for multiple ports could come in useful.

iii. The server could be made more rugged and reliable by making it fault-tolerant and by having it make automatic backups or replicates of files to guard against loss due to accidental destruction.

iv. The redirector could be modified to do local buffering of data for files opened in an exclusive (non-shareable) mode.

v. Instead of relying on MS-DOS, the server could be modified to run in an enhanced multitasking environment like XENIX, providing more sophisticated features and better resource management.

APPENDIX A

THE IIT-K PC-LAN

This section provides a brief introduction to the IIT-K PC-LAN from a user's point of view. More information can be obtained from the references [1] and [2].

A.1 Topology and connections

The IIT-K PC-LAN is a token ring lan supporting upto 255 nodes. The ring is unidirectional and currently operates at 192 Kbaud with twisted wire as the physical medium.

One of the nodes is designated as the primary node,some of its special functions being to initiate token circulation at power-on, to reintroduce the token in case of failure and to remove circulating packets. All other nodes are secondary nodes and are connected to the network through cluster modules.

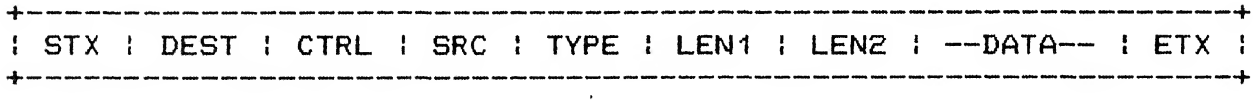
A cluster module can support a maximum of 4 PCs. Its function is to isolate the PCs from the network so that individual stations can arbitrarily come up or go down without affecting the activity on the network. This is achieved by the opto-couplers and multiplexing logic on each cluster module, which automatically bypass a node when the PC is off. The interface between a PC and a cluster module is through an Interface Message Processor (IMP) card which fits into one of the expansion slots on the PC. The cluster module connects to the ring through balanced current drivers/receivers (75110/75108).

A.2 Packet flow

The host PC writes the packet to be transmitted to its IMP card which stores it in one of two transmit buffers.When the node has successfully captured the circulating token, it transmits all pending packets and then forwards the token.A transmitted packet first reaches the primary node, which puts it into a repeat buffer and sends it forward. When the packet comes to the primary again, it is removed from the ring. The destination node receives the packet only when it is relayed by the primary in this manner. Packets which are not addressed to it and packets which are on their first trip to the primary are simply passed on. Upon receiving a valid packet,the destination node sends an acknowledgment to the source node.

A.3 Frame structure

The following is the frame structure used by the lan :



The SRC and DEST fields are 8-bit node addresses. The address 00h is reserved for the primary node, while 0ffh is used for broadcast packets. Thus the secondary nodes can have addresses between 01h and 0feh.

The CTRL field carries control information. Individual bits indicate whether the packet is going from the src to the primary or from the primary to the destination; whether the packet is a data packet or an acknowledge packet,etc.

The TYPE field provides information about the type of data that the packet contains. This will be of use to higher level protocols.

The LEN1,LEN2 fields contain the length of the data. (LEN1 is the higher byte.) In the current implementation LEN1 is always 0 since the maximum packet length supported is 255 bytes.

A.4 The IMP card and the host interface

The IMP card contains a CPU (8088), a MUART (8256), memory for the firmware and buffers, I/O ports for the PC interface and associated logic. Transmission of a packet by the PC is done in polled mode, while reception is in interrupt mode.

The lowest 3 layers of network protocol, viz. the physical,data link and network layers, are implemented on the IMP card. A simple stop & wait protocol is used for packet transmission : if the acknowledgment to a transmitted packet does not arrive in a fixed time, it is retransmitted. The hardware also takes care of parity generation and checking.

The IMP card uses 2 addresses in the I/O map of the host PC

340h - Data I/O port

348h - Control register (write)
- Status register (read)

Control and status register bits :

Bit 15 : TXBF - Transmit Buffer Full
Bit 14 : SPECL - Byte ready at data port is Special
Bit 13 : EXPB - Expecting the first byte of a packet
Bit 12 : IBE - Input buffer empty; ready to accept next byte
Bit 1 : PCMD - Treat the following bytes as commands rather than as data

Valid commands :

Request to send
Reset
Clear transmit buffer
Clear receive buffer

Before writing a packet to the card, the host PC must make sure that the card has an empty transmit buffer to hold the packet. This is done by checking the TXBF bit in the status register. If this is clear the host proceeds to write the actual command byte RTS (Request To Send) to the data port (by manipulating the PCMD bit in the control register). The card then responds by setting the EXPB bit in the status register, which indicates that it is ready to accept a packet from the host. Following this, the host writes the packet to the card in the order :

DEST, TYPE, LEN2, data

Before writing each byte, the host must wait till the IBE bit in the status register is set, indicating that the card is ready to accept the next byte.

The host receives a packet from the card in interrupt mode. The card generates an interrupt on the IRQ2 line for each byte of the packet. The interrupt service routine has to read a byte from the card and send an EOI to the 8259A interrupt controller. The packet is delivered in the following order :

NEW, DEST, CTRL, SRC, TYPE, LEN1, LEN2, data

The first byte is a special byte indicating that a new packet has been received.

APPENDIX B

NOTES

B.1 Initializing the server file system

The server expects the user/group/directory information to be available in the files USR.INF, GRP.INF and DIR.INF respectively. When installing the server these files must be created by running the program BUILDINF.EXE from the drive that is to contain the server file system. (Other drives can be included by JOINing them to this one).

The DIR.INF so created contains entries for all the existing directories on the drive. All of them are assumed to be owned by the superuser (user no.0); the 'rights', 'groups' and 'ispublic' fields are set to deny access to all other users.

USR.INF will contain just one user, the SUPERUSER, with no password and full rights. GRP.INF wii be initially empty.

Once the server is running (Sec. B.2), the superuser can login and use the appropriate options in the utilities to set his password and to add users and groups. When a new user is added, a directory of the same name (and owned by him) is created in the root. The user's rights and password must be set separately. Removing a user does not automatically remove his files/directories.

B.2 Setting up the server

The server program is in the file FS.EXE. Before running it, however, the following things have to be done :

i. The PC-LAN card must be properly installed.

ii. The driver NETDRVR.SYS must be installed, specifying the server internet address as parameter.

iii. The maximum number of handles must have been set at 20 by the 'FILES = ' line in CONFIG.SYS.

iv. The USR.INF, GRP.INF and DIR.INF files must be present in the current directory.

v. The DOS file sharing support module (SHARE.EXE) must be loaded.

When the server is ready to process requests it displays the message 'Ready....'. Commands can be entered by pressing the 'Esc' key and waiting for the server to respond with a '?' prompt. The following commands are supported currently :

'S' - Shutdown the server (asks for confirmation)

'C' - print details about the status of the Cache

'F' - Force closure of a file in the cache

B.3 Accessing the server

Logging in, defining network drives and logging out are all accomplished through the utilities (U.COM). Before attempting to log in,

i. The PC-LAN card must be installed.

ii. The driver NETDRVR.SYS must be installed, specifying the internet address of the local PC as parameter.

iii. The LASTDRIVE must have been set appropriately (through the 'LASTDRIVE = ' line in CONFIG.SYS). Only drives beyond the LASTDRIVE can be defined as network drives. Thus if the LASTDRIVE is set to 'Z', files on the server cannot be accessed.

iv. The redirector (RDR.COM) must be installed. (This could be included in the AUTOEXEC.BAT file for convenience).

B.4 Modifying and compiling

All the code for the system was developed using the Turbo C compiler (version 2.0) and the Turbo assembler (version 1.0). The source listing appears in Appendix C. The listing is not complete; some of the utility routines have been excluded due to shortage of space. The full source code is available at the MDS Lab, ACES, IIT Kanpur.

The source files relating to the server, the redirector, the utilities and the driver must be in separate directories (not necessarily in the root) : FS, RDR, U and NETDRVR respectively. All the dependencies are declared in the corresponding MAKEFILEs.

The redirector and the driver must be compiled in the 'tiny' model of Turboc and converted to .COM files.

When modifying the redirector it must be remembered that since it is a memory-resident program, the data and stack segments will not be the same when it receives control. But the compiler assumes that DS = SS. Thus when accessing a stack variable (automatic variables and parameters) through a pointer, a segment override (SS:) must be used. In Turbo C this can be

done by declaring the pointer to be an '_ss *' instead of a normal '*'. This problem does not arise in the driver because it uses stack-switching, which ensures that SS = DS. The redirector cannot use stack-switching because of function 4bh (EXEC) which requires the code to be re-entrant. The safest way to handle the problem is to put all such variables (those that will be accessed through pointers) in the data segment, by declaring them 'static'.

APPENDIX C

THE SOURCE LISTING

```
/* fs.h - the server handler function declarations */

#include "..\rdr\freq.h"

/* handlers corresponding to codes in 'freq.h' */

#define         FSHNDLRS        fsread,\
                                fswrite,\
                                fsopen,\
                                fsunlink,\
                                fsrename,\
                                fschdir,\
                                fsmkdir,\
                                fsrmdir,\
                                fssearch,\
                                fssearch,\
                                fscreat,\
                                fscreattemp,\
                                fscreatnew,\
                                fsmod,\
                                fsmod,\
                                fsdt,\
                                fsdt,\
                                fsflen,\
                                fsclose,\
                                fsspace,\
                                fslock,\
                                fsunlock,\
                                fsinfo
```

```
/* fac.h - declarations for access checking functions */

#include "..\rdr\freq.h"

/* The files used to save user/group/directory information */
#define USRFILE         "usr.inf"
#define GRPFILE         "grp.inf"
#define DIRFILE         "dir.inf"

/* some limits */
#define MAXUNAM 8                       /* maximum length of a user/group name */
#define MAXUSERS 8                      /* no. of users */
#define MAXGROUPS 8                     /* no. of groups */
#define MAXUGR 8                        /* no. of groups a user can allow to access a directory */
#define MAXGRM 11                       /* no. of members in a group */

#define LSIZE 256                       /* size of log table */
#define EMPTY 0xffff                    /* no user/group */
#define HSIZE 101                       /* hash table size */


typedef unsigned int    u_int;
```

```
/* the user/group/directory table entry structures */
typedef struct user
{
        char name [MAXUNAM];
        char pwd [MAXUNAM];
        u_int rights;
        u_int scn;                      /* starting cluster no. of user's 'home' directory */
        int tncl;                       /* total no. of clusters allocated to user */
        int free;                       /* total no. of free clusters */
        u_int reserved [4];             /* just to round off */
} USER;

typedef struct group
{
        char name [MAXUNAM];
        u_int rights;
        u_int mems [MAXGRM];
} GROUP;

typedef struct dir
{
        char name [40];
        u_int owner;
        u_int ispublic;
        u_int rights;
        u_int groups [MAXUGR];                  /* groups allowed access to this directory */
        struct dir *next;
} DIR;

struct req                                      /* information about current request */
{       u_int uno;                              /* user making the request */
        DIR *dirp;                              /* directory being accessed */
};


/* bits in user/group/directory rights */
#define         R_NONE  0x0000
#define         R_ALL   0x00ff
#define         R_READ  0x0080
#define         R_WRITE 0x0040
#define         R_OPEN  0x0020
#define         R_CREAT 0x0010
#define         R_DEL   0x0008
#define         R_SRCH  0x0004
#define         R_MOD   0x0002
#define         R_PAR   0x0001


#define user_has_right(uno,r)           ((utab [uno].rights & (r)) == (r))
#define group_has_right(gno,r)          ((gtab [gno].rights & (r)) == (r))
#define dir_right_defined(dirp,r)       (((dirp)->rights & (r)) == (r))
#define issup(uno)                      ((uno) == 0)
#define isself(u)                       ((u) == req.uno)
#define isowner(uno,dirp)               ((dirp)->owner == (uno))
```

```
u_int hash (char *s);
int access_chk (char *name, u_int uid, u_int op);
int ismember (u_int gno, u_int uno);
int fslogin (FREQ *p);
int fslogout (FREQ *p);
char *tab_init (void);
DIR *facmkdir (char *name);
int facrmdir (char *name);
USER *getusrentry (u_int uno);
```

```
/* fch.h - declarations for the server file cache */

#include "..\rdr\freq.h"

typedef struct fileinfo                 /* the cache entry */
{
        char name [MAXNAMLEN];
        int hndl;                       /* handle */
        unsigned uno;                   /* user who accessed it */
        int mode;                       /* access mdoe */
        long pos;                       /* file pointer position */
        int lock;                       /* no. of locks */
        struct fileinfo *next;
        struct fileinfo *prev;
} C_FILE;

#define HEAD    fcache
#define NEWEST  HEAD->prev
#define OLDEST  HEAD->next

/* the size of the cache */
#define MAXOFILES 8

#define isopen(p)       ((p)->hndl >= 0)
#define islocked(p)     ((p)->lock)
#define shared(mode)    (((mode) & 0x70) != 0)

/* convert the access mode from DOS format to the one used in Turbo C */
#define cnvrt(mode)     (ctab[(mode) & 0x3]) | ((mode) & ~0x3))

#define READMODE        (O_RDONLY | O_RDWR)
#define WRITEMODE       (O_WRONLY | O_RDWR)
#define ANYMODE         (O_RDONLY | O_WRONLY | O_RDWR)

extern int ctab[];      /* look-up table for cnvrt() */

char *cache_init (void);
C_FILE * c_open (char *name,int mode, unsigned uno);
int c_read (C_FILE *p, void *data, int n);
int c_write (C_FILE *p, void *data, int n);
int c_close (C_FILE *p);
int c_seek (C_FILE *p, long pos);
void uncache (char *name, unsigned uno);
```

```
int purge (char *name);
void printcache (void);
```

```
/* ds.h -  declarations associated with disk space */

struct boot
{
        unsigned char jmp [3];
        unsigned char oem_name [8];
        unsigned int bps;
        unsigned char spc;
        unsigned int nrs;
        unsigned char ncf;
        unsigned int nerd;
        unsigned int tns;
        unsigned char media_id;
        unsigned int spf;
        unsigned int spt;
        unsigned int nh;
        unsigned int nsrs;
        unsigned char resvd [482];
};

struct dp                       /* disk parameters */
{
        unsigned int tnrs;      /* total no. of reserved sectors */
        unsigned int bpc;       /* bytes per cluster */
        unsigned char spc;      /* sectors per cluster */
};


typedef struct direntry
{
        unsigned char name [8], ext [3];
        unsigned char attr;
        unsigned char resvd [10];
        unsigned time, date;
        unsigned sc_no;         /* starting cluster no. */
        unsigned long size;
} DIRENTRY;

struct ufcb                     /* unopened FCB returned by fns. 11h, 12h */
{       unsigned char drive;
        struct direntry d;
};

struct uxfcb                    /* unopened extended FCB */
{
        char flag;
        char resvd[5];
        char attr;
        struct ufcb ufcb;
};
```

```
#define NENTRIES                32
#define BYTES_PER_SECTOR        512
#define BUFSIZE         (NENTRIES * sizeof (DIRENTRY))
#define NSECT           (BUFSIZE / BYTES_PER_SECTOR)


#define invalid(ep)     ( *ep->name == 0xe5 || *ep->name == '.' )
#define isfile(ep)      ( (ep->attr & 0x18) == 0 )
#define isdir(ep)       ( ep->attr & 0x10 )

/* convert size in bytes to clusters */
#define tocl(bytes)     ( (unsigned) ((bytes)/d.bpc + ( ((bytes) & (d.bpc - 1))? 1 : 0 )) )

/* convert cluster no. to LSN */
#define tolsn(sc_no)    ( ((sc_no) - 2) * d.spc + d.lnrs )
```

```
/* fsio.h - driver interface declarations */

#define PORT    0               /* driver port to be used */

char *dev_init (int port);
FREQ *getreq (void);
void reply (FREQ *p, int len);
```

```
/* fsp.h - disk space function declarations */

unsigned getscn (char *dir);
int fsspace (FREQ *p);
int quota_chk (unsigned uno);
void getdp (void);
```

```
/* fserrno.h - special error codes for the server */

/* note: all these codes are negative, for easy integration with the DOS error codes */
enum sys_err_ops
{       EGTABFULL = -8,         /* cannot add group : Group table full */
        EMKUSERDIR,             /* unable to make directory for new user */
        EUTABFULL,              /* cannot add user : User table full */
        EQUOTA,                 /* Disk quota exceeded */
        EDRV,                   /* Invalid drive */
        EPORT,                  /* Unable to open driver port */
        ENLOGGED,               /* Not logged in */
        ELOGGED,                /* already logged in */
        OK                      /* = EZERO = 0; no error */
};
```

```
/* fs.c - server main program */

#include <stdio.h>
#include <stdlib.h>
```

```
#include <conio.h>
#include <ctype.h>
#include <bios.h>
#include <dos.h>
#include "fs.h"
#include "fac.h"
#include "fch.h"
#include "fsio.h"
#include "..\rdr\bio.h"

#define ESC     0x1b

char *fsinit (void);
void check_cmds (void);
int ctrlbrk_hndlr (void);
int harderr_hndlr (void);

main ()
{
        static int (*reqhndlr[]) (FREQ *p) =
        {
                FSHNDLRS
        };

        register FREQ *p;
        register unsigned op;
        int code;
        char *s;

        if ((s = fsinit ()) != NULL)                    /* initialization */
        {       puts (s);
                return 1;
        }

        while ((p = getreq ()) != NULL)                 /* wait for a request */
        {
                op = p->hdr.code;

                /* check access  and pass on to appropriate handler */

                p->hdr.code = ((code = access_chk (p->hdr.name, p->hdr.id, op)) != 0)? code
                            : (*reqhndlr [op < UTILS? op : UTILS]) (p);

                /* send reply */
                reply (p, (p->hdr.code)? FHDRLEN : (FHDRLEN + p->hdr.len));
        }
}


char *fsinit (void)
{
        char *s;

        if ((s = dev_init (PORT)) != NULL       /* initialize driver */
         || (s = tab_init ()) != NULL           /* user/group/dir tables */
```

```
        || (s = cache_init ()) != NULL)         /* cache */
                return s;

/*      setverify (1);*/
        setcbrk (0);                            /* disable control-break checking */
        ctrlbrk (ctrlbrk_hndlr);                /* control-break and */
        harderr (harderr_hndlr);                /* critical error handlers */

        puts ("Ready....");
        return NULL;
}


int ctrlbrk_hndlr (void)
{
        return 1;                       /* no action on control-break */
}


int harderr_hndlr (void)
{
        hardresume (3);                 /* fail system call on critical error */
}


void check_cmds (void)
{
        register int key;
        static int esc = 0;
        char s[MAXNAMLEN];

        while (bioskey (1) != 0)
        {
                if ((key = getch ()) == ESC)
                {       cputs ("\n\r?");
                        esc = 1;
                        continue;
                }

                if (!esc)
                        continue;

                switch (putch (toupper (key)))
                {
                        case 'S' :      cputs ("\n\rShutdown? ");
                                        if (toupper (getche ()) == 'Y')
                                                exit (0);
                                        break;

                        case 'C' :      cputs ("\n\rCache Status :");
                                        printcache ();
                                        break;

                        case 'F' :      cputs (" : File to close ? ");
                                        if (*gets (s) && purge (s) != 0)
```

```
                                        cputs ("File not found");
                                break;

                        default  :      cputs (" : Unknown command");
                                break;
                }

                cputs ("\n\rReady....\n\r");
                esc = 0;
        }
}
```

```
/* fac.c - access control */

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <dos.h>
#include <dir.h>
#include "..\rdr\freq.h"
#include "fserrno.h"
#include "fac.h"
#include "fsp.h"

/* decode user id to get user no.*/
#define user(uid)       (*(u_int *)(uid))

#define         INIT    0
#define         UPDATE  1
#define         SAVE    2

DIR *getdir (char *name, int isdir);
char *utab_io (int op, u_int uno);
char *gtab_io (int op, u_int gno);
char *dtab_io (int op);
void update (void);

/* the user, group, directory table pointers, space for them is allocated at run-time */
static USER *utab;
static GROUP *gtab;
static DIR **dtab;

static u_int *logtab;

/* rights for each code defined in 'freq.h' */
static u_int rtab [] =
{
        R_READ,
        R_WRITE,
        R_OPEN,
        R_DEL,
        R_DEL,          /* RENAME */
        R_NONE,
```

```
        R_PAR,
        R_PAR,
        R_SRCH,
        R_SRCH,
        R_CREAT,
        R_CREAT,
        R_CREAT,
        R_NONE,
        R_NONE,
        R_NONE,
        R_NONE,
        R_NONE,
        R_NONE,
        R_NONE,
        R_NONE,
        R_NONE,
        R_NONE,
        R_NONE
};


/* info about the user making the request and the directory being accessed */
struct req req;

/* the main access checking routine */

int access_chk (char *name, u_int uid, u_int op)
{
        u_int *g;
        register DIR *p;

        /* right required for the requested operation */
        register u_int r = op >= UTILS? R_NONE : rtab[op];

        req.uno = EMPTY;
        req.dirp = NULL;

        if (op == UTILS + LOGIN)                /* LOGIN is free for all */
                return OK;

        if ((req.uno = user (uid)) == EMPTY)    /* check if user has logged in all right */
                return EACCES;

        if (!user_has_right (req.uno,r))        /* check user rights */
                return EACCES;

        if (*name == '\0')                      /* no file/dir name mentioned */
                return OK;

        if ((req.dirp = p = getdir (strupr (name),(op == CHDIR || op >= UTILS))) == NULL)
                return ENOPATH;                 /* check if directory is known to server */

        if (issup (req.uno) || isowner (req.uno, p))
                return OK;                      /* superuser and owner are allowed access */
```

```
        if (!dir_right_defined (p,r))
                return EACCES;                  /*  for others, the dir. right must be set */

        if (p->ispublic)
                return OK;                      /* anyone can access a public dir. */

        /* check if access can be allowed by virtue of group membership */
        for (g = p->groups; g < &p->groups[MAXUGR]; g++)
                if (*g != EMPTY && group_has_right (*g, r)
                                && ismember (*g, req.uno))
                        return OK;

        return EACCES;                  /* all cases exhausted, deny access */

}


/* return a pointer to entry in the dir.info table */
static DIR *getdir (register char *name, int isdir)
{
        char *s = NULL;
        register DIR *p;

        /* if the name refers to a file,get the parent directory */
        if (!isdir && (s = strrchr (name,'\\')) != NULL)
                *s = '\0';

        if (s == name)
                name = "\\";            /* the root! */

        /* index into the hash table and search the linked list */
        for (p = dtab [hash (name)]; p != NULL; p = p->next)
                if (strcmp (p->name,name) == 0)
                        break;

        if (s)
                *s = '\\';

        return p;
}


/* check if user 'uno' is a member of group 'gno' */
int ismember (u_int gno, u_int uno)
{
        register GROUP *g = &gtab[gno];
        register u_int *m;

        if (*g->name == '\0')
                return 0;
        for (m = g->mems; m < &g->mems[MAXGRM]; m++)
                if (*m == uno)
                        return 1;
        return 0;
}
```

```
/* the hash function for the dir. table search */
u_int hash (register char *s)
{
        register u_int n = 0;

        while (*s)
                n += *s++;

        return (n % HSIZE);
}


/* server login */
int fslogin (FREQ *p)
{
        int i;
        u_int uno = p->hdr.par1;

        /* check if user is legal */
        if (uno > MAXUSERS || *utab[uno].name == '\0'
                        || strcmp (p->hdr.name, utab [uno].pwd) != 0)
                return EACCES;
        if (quota_chk (uno) != 0)               /* check if disk quota is OK */
                return EQUOTA;

        /* allocate a random entry in logtab */
        while (logtab [(i = random (LSIZE))] != EMPTY)
                ;

        logtab [i] = uno;
        p->hdr.par1 = (u_int) &logtab [i];      /* return a pointer to it as the id */
        return OK;
}


/* server logout */
int fslogout (FREQ *p)
{
        u_int uno;

        if ((uno = user (p->hdr.id)) == EMPTY)
                return OK;

        if (quota_chk (uno) != 0)
                return EQUOTA;

        user (p->hdr.id) = EMPTY;       /* reset logtab entry */
        return OK;
}
```

```
/* initialize all the tables */
char *tab_init (void)
{
        u_int *l;
        char *s;

/* allocate memory for them first */
        if ((dtab = (DIR **) calloc (HSIZE, sizeof (DIR *))) == NULL
           || (logtab = (u_int *) calloc (LSIZE, sizeof (int))) == NULL
           || (utab = (USER *) calloc (MAXUSERS, sizeof (USER))) == NULL
           || (gtab = (GROUP *) calloc (MAXGROUPS, sizeof (GROUP))) == NULL)
                return ("Not enough memory");

        for (l = logtab; l < &logtab [LSIZE]; l++)
                *l = EMPTY;

/* fill them up from the corresponding files */
        if ((s = utab_io (INIT, 0)) != NULL
         || (s = gtab_io (INIT, 0)) != NULL
         || (s = dtab_io (INIT)) != NULL)
                return s;

        chdir ("\\");           /*  ensure that the current dir. is the root */

        getdp ();               /* read in disk parameters */
        return NULL;
}


/* directory table manager */
char *dtab_io (int op)
{
        static FILE *fp;
        static int changes = 0;
        register DIR *p;
        register int i;
        DIR **d;

        switch (op)
        {       case INIT :     if ((fp = fopen (DIRFILE, "r+b")) == NULL)
                                        return ("Unable to open dir. file");

                                for (;;)
                                {       if ((p = (DIR *) malloc (sizeof (DIR))) == NULL)
                                                return ("Not enogh memory");

        /* read in an entry */                  if (fread ((void *) p, sizeof (DIR), 1, fp) == 0)
                                                break;

        /* install it in the table */   p->next = dtab [i = hash (p->name)];
                                        dtab [i] = p;
                                }

                                free (p);
                                atexit (update);        /* file must be updated on exit  */
```

```
                        break;

                case UPDATE :   changes = 1;
                                break;

                case SAVE :     if (!changes)           /* write info back to disk */
                                        break;
                                changes = 0;
                                fseek (fp, 0, SEEK_SET);
                                for (d = dtab; d < &dtab[HSIZE]; d++)
                                        for (p = *d; p != NULL; p = p->next)
                                                fwrite (p, sizeof (DIR), 1, fp);
                                break;
        }

        return NULL;
}


void update (void)
{
        dtab_io (SAVE);
}


/* user table manager */
char *utab_io (int op, u_int uno)
{
        static FILE *fp;

        switch (op)
        {       case INIT :     if ((fp = fopen (USRFILE, "r+b")) == NULL)
                                        return ("Unable to open user file");

                                fread (utab, sizeof (USER), MAXUSERS, fp);
                                break;

                case UPDATE :   if (fseek (fp, uno * sizeof (USER), SEEK_SET) != 0
                                 || fwrite (&utab[uno], sizeof (USER), 1, fp) != 1)
                                        printf ("Error updating user %d", uno);
                                break;

        }

        return NULL;

}


/* group table manager */
char *gtab_io (int op, u_int gno)
{
        static FILE *fp;
```

```
        switch (op)
        {       case INIT :     if ((fp = fopen (GRPFILE, "r+b")) == NULL)
                                        return ("Unable to open group file");

                                fread (gtab, sizeof (GROUP), MAXGROUPS, fp);
                                break;

                case UPDATE :   if (fseek (fp, gno * sizeof (GROUP), SEEK_SET) != 0
                                 || fwrite (&gtab[gno], sizeof (GROUP), 1, fp) != 1)
                                        printf ("Error updating group %d",gno);
                                 break;
        }

        return NULL;

}


/* make a new entry in the dir. table */
DIR *facmkdir (char *name)
{
        register DIR *p;
        int i;
        register u_int *g;

        if ((p = (DIR *) malloc (sizeof (DIR))) == NULL)
                return NULL;

        strcpy (p->name,name);
        p->owner = req.dirp->owner;             /* belongs to user owning the parent */
        p->rights = R_ALL;
        p->ispublic = 0;

        for (g = p->groups; g < &p->groups[MAXUGR]; g++)
                *g = EMPTY;

        p->next = dtab [i = hash (name)];       /* install in table */
        dtab [i] = p;

        dtab_io (UPDATE);                       /* update info */
        return p;
}


/* remove an entry from the dir. table */
int facrmdir (char *name)
{
        int i = hash (name);
        register DIR *d;
        register DIR *p = dtab[i];

        if ((d = getdir (name, 1)) == NULL)
                return ENOPATH;                 /* no such entry */
```

```
        if (p == d)
                dtab[i] = d->next;

        else
        {       while (p != NULL && p->next != d)
                        p = p->next;            /* search the linked list */

                if (p == NULL)
                        return ENOPATH;

                p->next = d->next;              /* remove from list */
        }

        free (d);
        dtab_io (UPDATE);
        return OK;
}


/* return a pointer to a user's info in the table */
USER *getusrentry (u_int uno)
{
        return &utab[uno];
}


/* handler for GETUSRINFO : return user table entry */
int getusrinfo (FREQ *p)
{
        if (p->hdr.par1 >= MAXUSERS)    /* par1 contains the user no. */
                return EINVFNC;

        if ( !(issup (req.uno) || isself (p->hdr.par1)) )
                return EACCES;

        *( (USER *)p->data ) = utab[p->hdr.par1];
        p->hdr.len = sizeof (USER);

        return OK;
}

/* handler for SETUSRINFO : update user table entry */
int setusrinfo (FREQ *p)
{
        if (p->hdr.par1 >= MAXUSERS)
                return EINVFNC;

        if (issup (req.uno))
                utab[p->hdr.par1] = *( (USER *) p->data);
        else if (isself (p->hdr.par1))
                memcpy (utab[p->hdr.par1].pwd ,((USER *)p->data)->pwd, MAXUNAM);
        else
                return EACCES;

        utab_io (UPDATE, p->hdr.par1);
```

```
        return OK;

}

/* handler for GETGRPINFO */
int getgrpinfo (FREQ *p)
{
        if (p->hdr.par1 >= MAXGROUPS)
                return EINVFNC;

        if ( !(issup (req.uno) || ismember (p->hdr.par1, req.uno)) )
                return EACCES;

        *((GROUP *) p->data) = gtab[p->hdr.par1];
        p->hdr.len = sizeof (GROUP);

        return OK;
}

/* handler for SETGRPINFO */
int setgrpinfo (FREQ *p)
{
        if (p->hdr.par1 >= MAXGROUPS)
                return EINVFNC;

        if (!issup (req.uno))
                return EACCES;

        gtab[p->hdr.par1] = *((GROUP *) p->data);

        gtab_io (UPDATE, p->hdr.par1);
        return OK;
}


int getdirinfo (FREQ *p)
{
        *((DIR *) p->data) = *req.dirp;
        p->hdr.len = sizeof (DIR);
        return OK;
}


int setdirinfo (FREQ *p)
{
        if (issup (req.uno))
                *req.dirp = *((DIR *) p->data);
        else if (isowner (req.uno, req.dirp))
        {       memcpy (req.dirp->groups, ((DIR *) p->data)->groups, MAXUGR);
                req.dirp->ispublic = ((DIR *) p->data)->ispublic;
                req.dirp->rights = ((DIR *) p->data)->rights;
        }
        else
                return EACCES;
```

```
        dtab_io (UPDATE);
        return OK;
}

/* handler for GETUSRLIST */
int getusrlist (FREQ *p)
{
        register USER *u = utab;
        register char (*n) [MAXUNAM] = (char (*)[MAXUNAM])p->data;

        while (u < &utab[MAXUSERS])
                memcpy (*n++, (u++)->name, MAXUNAM);

        p->hdr.len = MAXUSERS * MAXUNAM;

        return OK;
}

int getgrplist (FREQ *p)
{
        register GROUP *g = gtab;
        register char (*n) [MAXUNAM] = (char (*)[MAXUNAM])p->data;

        while (g < &gtab[MAXGROUPS])
                memcpy (*n++, (g++)->name, MAXUNAM);

        p->hdr.len = MAXGROUPS * MAXUNAM;

        return OK;
}


int getugrps (FREQ *p)
{
        u_int *d = (u_int *)p->data;
        register int i;

        if (p->hdr.par1 >= MAXUSERS)
                return EINVFNC;

        if ( !(issup (req.uno) || isself (p->hdr.par1)) )
                return EACCES;

        for (i = 0; i < MAXUGR; i++)
                if (ismember (i, p->hdr.par1))
                        *d++ = i;

        *d++ = EMPTY;
        p->hdr.len = (char *)d - p->data;

        return OK;
}
```

```
/* handler for MKUSER : install a new user */
int mkuser (FREQ *p)
{
        register USER *u;
        DIR *d;
        register char name[MAXUNAM + 1] = "\\";

        if (!(issup (req.uno)))
                return EACCES   ;                       /*only super user can do this */

        for (u = utab; u < &utab[MAXUSERS]; u++)        /* already exists */
                if (strcmp (u->name, strupr (p->data)) == 0)
                        return (p->hdr.par1 = u - utab, OK);

        for (u = utab; u < &utab[MAXUSERS] && *u->name; u++)
                ;
        if (u >= &utab[MAXUSERS])                       /* find empty user table entry */
                return EUTABFULL;

        strncpy (name + 1, p->data, MAXUNAM);
        if (mkdir (name) < 0  ||  (d = facmkdir (name)) == NULL)        /*make dir.for user */
                return EMKUSERDIR;

        strcpy (u->name, name + 1);
        *u->pwd = '\0';
        u->rights = R_NONE;
        u->scn = getscn (name + 1);                     /* get starting cluster no. of dir. */
        u->incl = 1;
        u->free = 0;

        utab_io (UPDATE, (p->hdr.par1 = d->owner = u - utab));
        return OK;
}

/* handler for RMUSER */
int rmuser (FREQ *p)
{
        if (!(issup (req.uno)))
                return EACCES;
        if (p->hdr.par1 >= MAXUSERS)
                return EINVFNC;

        *utab[p->hdr.par1].name = '\0';                 /* erase user name */
        utab_io (UPDATE, p->hdr.par1);
        return OK;
}


/* handler for MKGROUP */
int mkgroup (FREQ *p)
{
        register GROUP *g;
        register u_int *m;
```

```
        if (!(issup (req.uno)))
                return EACCES;

        for (g = glab; g < &glab[MAXGROUPS]; g++)       /* group already exists /*
                if (strcmp (g->name, strupr (p->data)) == 0)
                        return (p->hdr.par1 = g - glab, OK);

        for (g = glab; g < &glab[MAXGROUPS] && *g->name; g++)
                ;
        if (g >= &glab[MAXGROUPS])                      /* find empty entry in table */
                return EGTABFULL;

        strcpy (g->name, p->data);
        g->rights = R_NONE;
        for (m = g->mems; m < &g->mems[MAXGRM]; m++)
                *m = EMPTY;

        glab_io (UPDATE, (p->hdr.par1 = g - glab));
        return OK;
}


/* handler for RMGROUP */
int rmgroup (FREQ *p)
{
        if (!(issup (req.uno)))
                return EACCES;
        if (p->hdr.par1 >= MAXGROUPS)
                return EINVFNC;

        *glab[p->hdr.par1].name = '\0';
        glab_io (UPDATE, p->hdr.par1);

        return OK;
}
```

```
/* frh.c - file access request handlers */

#include <stdio.h>
#include <io.h>
#include <fcntl.h>
#include <string.h>
#include <dos.h>
#include <dir.h>
#include <errno.h>
#include "fserrno.h"
#include "..\rdr\freq.h"
#include "fch.h"
#include "fac.h"
#include "finf.h"
#include "..\rdr\str.h"


int illfunc (FREQ *p);
```

```
struct sfcb                     /* special FCB */
{
        char fcb_drive;
        char fcb_name [8]; char fcb_ext [3];
        char resvd [5];
        char sfcb_name [8]; char sfcb_ext [3];
};


static struct fcb f;
extern struct req req;          /* info about user making request, dir. being accessed */


/* handler for OPEN */
int fsopen (register FREQ *p)
{
        return( (c_open (p->hdr.name, cnvrt (p->hdr.F_MODE), req.uno) == NULL)? errno : OK );
}


/* CLOSE */
int fsclose (FREQ *p)
{
        uncache (p->hdr.name, req.uno);
        return OK;
}


/* READ */
int fsread (register FREQ *p)
{
        C_FILE *cp;
        register int n;

        if ((n = p->hdr.F_CNT) > FMAXDAT)
                n = FMAXDAT;

        if ((cp = c_open (p->hdr.name, READMODE, req.uno)) == NULL      /* open the file */
                || c_seek (cp, p->hdr.pos) < 0                          /* seek to required position */
                || (n = c_read (cp, p->data, n)) < 0)                   /* read */

                return errno;

        p->hdr.pos += (p->hdr.len = p->hdr.F_CNT = n);  /* return no. of bytes read and new position */
        return OK;
}


/* WRITE */
int fswrite (register FREQ *p)
{
        C_FILE *cp;
        register int n;
```

```
        if ((n = p->hdr.F_CNT) > FMAXDAT)
                n = FMAXDAT;

        if ((cp = c_open (p->hdr.name, WRITEMODE, req.uno)) == NULL
                || c_seek (cp, p->hdr.pos) < 0
                || (n = c_write (cp, p->data, n)) < 0)

                return errno;

        p->hdr.pos += (p->hdr.F_CNT = n);
        p->hdr.len = 0;

        return OK;
}


/* DELETE */
int fsunlink (register FREQ *p)
{
        register char *s;

        if ( unlink (p->hdr.name) == 0 )
                return (uncache (p->hdr.name, req.uno), OK);

        if (errno == EACCES)
                return EACCES;

        if (strrchr (p->hdr.name, '?') == NULL)         /* wildcards ? */
                return ENOFILE;

        *( s = strrchr (p->hdr.name, '\\')) = '\0';
        chdir (p->hdr.name);

        parsfnm (++s, &f, 0);                           /* delete using FCB call */
        _DX = (unsigned) &f;
        _AH = 0x13;
        geninterrupt (0x21);
        chdir ("\\");

        return ( (_AL == 0)? OK : ENOFILE);
}


/* RENAME */
int fsrename (register FREQ *p)
{
        register char *s;

        if (rename (p->hdr.name, p->data) == 0)
                return (uncache (p->hdr.name, req.uno), OK);
        if (errno == EACCES)
                return EACCES;

        if (strrchr (p->hdr.name, '?') == NULL  &&  strrchr (p->data, '?') == NULL)
                return ENOFILE;                                 /* wildcards ? */
```

```
        *(s = strrchr (p->hdr.name, '\\')) = '\0';
        chdir (p->hdr.name);

        parsfnm (++s, &f, 0);
        parsfnm (strrchr (p->data, '\\') + 1, (struct fcb *)(((struct sfcb *) &f)->sfcb_name - 1), 0);

        _DX = (unsigned) &f;
        _AH = 0x17;
        geninterrupt (0x21);                                    /* rename with FCB call */
        chdir ("\\");

        return ( (_AL == 0)? OK : ENOFILE);
}


/* CHDIR */
int fschdir ()
{
        return OK;
}


/* MKDIR */
int fsmkdir (FREQ *p)
{
        return ( (mkdir (p->hdr.name) < 0)? errno
                : (facmkdir (p->hdr.name) == NULL)? EACCES      /* update dir. table */
                : OK );
}


/* RMDIR */
int fsrmdir (FREQ *p)
{
        return ( (rmdir (p->hdr.name) < 0)? errno
                :facrmdir (p->hdr.name) );                      /* update dir. table */
}


/* SEARCHF, SEARCHN */
int fssearch (FREQ *p)
{
        struct ffblk buf;
        register struct ffblk *s = (struct ffblk *) p->data;
        register int i = 0;

        buf = *s;

        switch (p->hdr.code)
        {
                case SEARCHF :  if (findfirst (p->hdr.name, &buf, p->hdr.F_ATTR) < 0)
                                        break;
                                *s++ = buf;
                                i++;                            /* fall through */
```

```
                case SEARCHN :  while ( i < p->hdr.F_NBUF  &&  findnext (&buf) == 0)
                                        *s++ = buf, i++;        /* return as many files as possible */
        }

        return ( (p->hdr.len = (int)(p->hdr.F_NBUF = i) * sizeof (struct ffblk))? 0 : ENOFILE );
}


/* CREAT */
int fscreat (register FREQ *p)
{
        register int fd;

        if ((fd = _creat (p->hdr.name, p->hdr.F_ATTR)) < 0)
                return errno;

        _close (fd);
        c_open (p->hdr.name, O_RDWR, req.uno);          /* enter into cache */
        return OK;
}


/* CREATNEW */
int fscreatnew (register FREQ *p)
{
        register int fd;

        if ((fd = creatnew (p->hdr.name, p->hdr.F_ATTR)) < 0)
                return errno;

        close (fd);
        c_open (p->hdr.name, O_RDWR, req.uno);
        return OK;
}


/* CREATTEMP */
int fscreattemp (register FREQ *p)
{
        register int fd;

        if ((fd = creattemp (p->hdr.name, p->hdr.F_ATTR)) < 0)
                return errno;

        close (fd);
        c_open (p->hdr.name, O_RDWR, req.uno);
        return OK;
}


/* CHMOD, GETMOD */
int fsmod (register FREQ *p)
{
        register int attrib;
```

```
        if ((attrib = _chmod (p->hdr.name, (p->hdr.code == CHMOD)? 1 : 0,
                        p->hdr.F_ATTR)) < 0)
                return errno;

        p->hdr.F_ATTR = attrib;
        return OK;
}

/* GETDATE, SETDATE */
int fsdt (FREQ *p)
{
        register C_FILE *cp;

        if ( (cp = c_open (p->hdr.name, ANYMODE, req.uno)) == 0
                || ((p->hdr.code == SETDT)? setftime (cp->hndl, (struct ftime *) p->data)
                                          : getftime (cp->hndl, (struct ftime *) p->data)) < 0 )
                return errno;

        return OK;
}

/* FLEN : return file length */
int fsflen (FREQ *p)
{
        register C_FILE *cp;

        if ( (cp = c_open (p->hdr.name, ANYMODE, req.uno)) == 0
                || (p->hdr.pos = filelength (cp->hndl)) < 0)

                return errno;
        return OK;
}

/* LOCK */
int fslock (FREQ *p)
{
        register C_FILE *cp;

        p->hdr.len = 0;
        if ((cp = c_open (p->hdr.name, ANYMODE, req.uno)) == NULL
           || lock (cp->hndl, p->hdr.pos, *(long *) p->data) < 0)
                return errno;

        cp->lock++;
        return OK;
}

/* UNLOCK */
int fsunlock (FREQ *p)
{
        register C_FILE *cp;

        p->hdr.len = 0;
        if ((cp = c_open (p->hdr.name, ANYMODE, req.uno)) == NULL
```

```
        || unlock (cp->hndl, p->hdr.pos, *(long *) p->data) < 0)
            return errno;

    cp->lock--;
    return OK;
}

/* handlers for UTILS */
int fsinfo (FREQ *p)
{
    register int func = p->hdr.code - UTILS;

    static int (*info_handlers []) (FREQ *p) =
    {
        illfunc,
        illfunc,
        fslogin,
        fslogout,
        illfunc,
        illfunc,
        illfunc,
        illfunc,
        getusrinfo,
        setusrinfo,
        getgrpinfo,
        setgrpinfo,
        getdirinfo,
        setdirinfo,
        getusrlist,
        getgrplist,
        getugrps,
        mkuser,
        rmuser,
        mkgroup,
        rmgroup
    };

    return ( func >= sizeof (info_handlers)/sizeof (int (*)())? EINVFNC
        : (*info_handlers [func]) (p) );
}


int illfunc ()
{
    return EINVFNC;
}
```

```
/* fch.c - the file cache and associated routines */

#include <stdio.h>
#include <io.h>
#include <string.h>
#include <alloc.h>
#include <dos.h>
```

```
#include <fcntl.h>
#include "fch.h"

void promote (C_FILE *);
void demote (C_FILE *);
C_FILE *getfree (void);

C_FILE *fcache;
int ctab[4] = { O_RDONLY, O_WRONLY, O_RDWR, ANYMODE };  /* look-up table for cnvrt() */


/* allocate memory for the cache and set up the doubly linked circular list */
char *cache_init (void)
{
        register C_FILE *p;

        if ((fcache = (C_FILE *) calloc (MAXOFILES + 1, sizeof (C_FILE))) == NULL)
                return ("Not enough memory");

        for (p = HEAD; p != &fcache [MAXOFILES + 1]; p++)
                p->hndl = -1, p->prev = p - 1, p->next = p + 1;

        ((--p)->next = HEAD)->prev = p;

        return NULL;
}


/* open a file - specify name, mode, user no. */
C_FILE * c_open (char *name, register int mode, unsigned uno)
{
        register C_FILE *p;
        int hndl;

        for (p = NEWEST; isopen (p); p = p->prev)              /* cache hit ? */
                if (p->uno == uno
                   && shared (mode)? p->mode == mode : (p->mode & 0x7) & mode
                   && strcmp (p->name,name) == 0)
                        break;

        if (!isopen (p))                                       /* no - open it and make a new entry */
        {
                if (mode == ANYMODE || mode == READMODE)
                        mode = O_RDONLY;
                else if (mode == WRITEMODE)
                        mode = O_WRONLY;

                /* get a free cache slot and try to open the file */
                if ((p = getfree ()) == NULL || (hndl = _open (name, mode)) < 0)
                        return NULL;

                p->hndl = hndl;
                p->uno = uno;                          /* fill in file info */
                p->pos = 0L;
                p->mode = mode;
```

```
                p->lock = shared (mode);
                strcpy (p->name,name);                  /* enter into cache */
        }

        promote (p);                                    /* make this file the NEWEST */
        return (p);
}


/* close a file */
int c_close (register C_FILE *p)
{
        if (_close (p->hndl) < 0)                       /* system call to close it */
                return -1;

        p->hndl = -1;
        p->lock = 0;
        demote (p);                                     /* make it the OLDEST */
        return 0;
}

/* promote a file to the NEWEST */
static void promote (register C_FILE *p)
{
        if (p == NEWEST)
                return;

        (p->prev->next = p->next)->prev = p->prev;      /* unlink p */

        (p->prev = NEWEST)->next = p;                   /* link to NEWEST */

        (NEWEST = p)->next = HEAD;                      /* link to head */
}

/* demote a file to the OLDEST */
static void demote (register C_FILE *p)
{
        if (p == OLDEST)
                return;

        (p->prev->next = p->next)->prev = p->prev;      /* unlink p */

        (p->next = OLDEST)->prev = p;                   /* link to OLDEST */

        (OLDEST = p)->prev = HEAD;                      /* link to head */
}


/* seek in a file */
int c_seek (register C_FILE *p,long pos)
{
        if (p->pos == pos)                              /* position correct */
                return 0;

        if ((pos = lseek (p->hndl, pos, SEEK_SET)) < 0) /* set position */
```

```
        if (!dir_right_defined (p,r))
                return EACCES;                  /*  for others, the dir. right must be set */

        if (p->ispublic)
                return OK;                      /* anyone can access a public dir. */

        /* check if access can be allowed by virtue of group membership */
        for (g = p->groups; g < &p->groups[MAXUGR]; g++)
                if (*g != EMPTY && group_has_right (*g, r)
                                && ismember (*g, req.uno))
                        return OK;

        return EACCES;                  /* all cases exhausted, deny access */

}


/* return a pointer to entry in the dir.info table */
static DIR *getdir (register char *name, int isdir)
{
        char *s = NULL;
        register DIR *p;

        /* if the name refers to a file,get the parent directory */
        if (!isdir && (s = strrchr (name,'\\')) != NULL)
                *s = '\0';

        if (s == name)
                name = "\\";            /* the root! */

        /* index into the hash table and search the linked list */
        for (p = dtab [hash (name)]; p != NULL; p = p->next)
                if (strcmp (p->name,name) == 0)
                        break;

        if (s)
                *s = '\\';

        return p;
}


/* check if user 'uno' is a member of group 'gno' */
int ismember (u_int gno, u_int uno)
{
        register GROUP *g = &gtab[gno];
        register u_int *m;

        if (*g->name == '\0')
                return 0;
        for (m = g->mems; m < &g->mems[MAXGRM]; m++)
                if (*m == uno)
                        return 1;
        return 0;
}
```

```
        while (isopen (p) && islocked (p))
                p = p->next;                    /* find a file that is not locked */

        if (isopen (p))
                c_close (p);

        return ((p == HEAD)? NULL : p);         /* should never touch the HEAD */
}

/* print cache status */
void printcache (void)
{
        register C_FILE *p;

        for (p = NEWEST; isopen (p); p = p->prev)
                printf ("%s:\tuser %d\tmode %d\t%s\n",
                        p->name, p->uno, p->mode, islocked (p)? "LOCKED" : "");

        for (; p != HEAD; p = p->prev)
                printf ("----------\n");
}
```

```
/* fsio.c - driver interface */

#include <io.h>
#include <fcntl.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include "..\rdr\freq.h"
#include "..\netdrvr\dev.h"
#include <bios.h>

int shutdown (void);
void close_dev (void);

static struct cblk cblk;
static int dev, devcfg;

char *dev_init (int port)               /* initialize driver */
{
        cblk.lport = port;
        if ((dev = _open (DEV, O_RDWR | O_BINARY)) < 0  /* open in BINARY mode */
            || ioctl (dev, 2, &cblk, 0) < 0)            /* open port */
                return ("Unable to open driver");

        if (cblk.lport != port)
                return ("Unable to open port");

        cblk.ack = 0;
        devcfg = ioctl (dev, 0);
        atexit (close_dev);                             /* port must be closed on exit */
```

```
        return NULL;
}


/* close port and driver */
void close_dev (void)
{
        ioctl (dev, 3, &cblk, 0);
        _close (dev);
}


/* get a request from a remote client */
FREQ *getreq (void)
{
        static FREQ req;
        cblk.rbuf = &req;

        while (!shutdown ())
        {       if (_read (dev, &cblk, FREQLEN) != 0)
                        return &req;
                ioctl (dev, 1, 0xe0);           /* reset EOF bit */
        }
        return NULL;
}



/* send a reply to the remote client */
void reply (FREQ *p, int len)
{
        cblk.xbuf = p;
        _write (dev, &cblk, len);
}



int shutdown (void)
{

        if (!bioskey (2))
                return 0;

        printf ("\nShutdown ? ");
        return ( (getche () & 0x5f) == 'Y' );

}
```

```
/* fsp.c - disk space routines */

#include <alloc.h>
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
```

```
#include "..\rdr\freq.h"
#include "fac.h"
#include "ds.h"
#include "fserrno.h"

static struct dp d;
extern struct req req;

/* get the number of clusters used up by a directory and all its children */
/* specify the starting cluster no. of the directory */
int getclusters (unsigned scn)
{
        DIRENTRY *buf;
        register DIRENTRY *ep;
        register unsigned nc = 0;
        unsigned lsn = tolsn (scn);             /* get corresponding LSN */

        if ( (buf = malloc (BUFSIZE)) == NULL)
                return 1;

        /* read in the directory sectors into the buffer */
        for (absread (2,NSECT,lsn,ep = buf);
             *ep->name;
             (ep < (buf + 31))? (int) ep++
                              : (absread (2,NSECT,lsn += NSECT,ep = buf)))

        {
                if (invalid (ep))
                        continue;
                nc += (isfile (ep))? tocl (ep->size)            /* for a file, add to sum */
                    : (isdir (ep))? getclusters (ep->sc_no)     /* for a dir., do a recursive call */
                    : 0;
        }

        free (buf);
        return nc + 1;  /* no. of clusters under this dir. + the cluster holding the dir.itself */
}


/* read in disk parameters */
void getdp (void)
{
        struct boot b;

        absread (2, 1, 0, &b);

        d.spc = b.spc;
        d.bpc = b.spc * 512;
        d.inrs = b.nrs +
                 (b.spf * b.ncf) +
                 (b.nerd * 32 / 512);
}

/* get starting cluster no. of a dir. */
unsigned getscn (char *dir)
```

```
{
        struct uxfcb f;
        static char dta [128];

        setdta ((char far *) &dta);
        parsfnm (dir, (struct fcb *) &f.ufcb,0);

        f.flag = -1;
        f.attr = 0x10;
        _AH = 0x11;             /* search for entry using fn.11h */
        _DX = (unsigned) &f;
        __int__ (0x21);

        return ((_AL == 0xff)? -1 : ((struct uxfcb far *) dta)->ufcb.d.sc_no);
}


/* handler for GETSPACE */
int fsspace (FREQ *p)
{
        USER *u;

        if (issup (req.uno))

                /* for the superuser, return info about the whole disk */
                getdfree ( 0,(struct dfree *) p->data);
        else
        {
                /* for a normal user, return info from his point of view */
                u = getusrentry (req.uno);
                ((struct dfree *) p->data)->df_avail = (u->free = u->tncl - getclusters (u->scn)) < 0?
                                                        0 : u->free;
                ((struct dfree *) p->data)->df_sclus = d.spc;
                ((struct dfree *) p->data)->df_bsec = 512;
                ((struct dfree *) p->data)->df_total = u->tncl;
        }

        p->hdr.len = sizeof (struct dfree);
        return OK;
}


/* check space left */
int quota_chk (u_int uno)
{
        USER *u;
        struct dfree dfree;

        if (issup (uno))
        {       getdfree (0, &dfree);
                getusrentry (0)->free = dfree.df_avail;
                return 0;
        }

        /* check space left and update user table */
```

```
        u = getusrentry (uno);
        return  (u->free = u->incl - getclusters (u->scn)) < 0;
}
```

```
/* buildinf.c - separate utility to create user, group and directory tables from scratch */

#include <stdio.h>
#include <stdlib.h>
#include <dir.h>
#include <dos.h>
#include <alloc.h>
#include <string.h>
#include "fac.h"


void treewrite (FILE *fp, char *path);
DIR * dirlist (char *path);

main ()
{
        FILE *fp;

        static USER superuser =
        {       "SUPER",
                "",
                R_ALL,
                0,0,0
        };

        struct dfree dfree;

        static DIR root =
        {       "\\",
                0,
                0,
                R_NONE
        };
        u_int *g;


        /* create user file with superuser as the only user */

        if ((fp = fopen (USRFILE, "wb")) == NULL)
                perror (USRFILE), exit (1);
        getdfree (0, &dfree);
        superuser.incl = dfree.df_total;
        superuser.free = dfree.df_avail;
        fwrite (&superuser, sizeof (USER), 1, fp);

        /* create group file (initially empty) */

        if (fopen (GRPFILE, "wb") == NULL)
                perror (GRPFILE), exit (1);
```

```
        /* create directory info file */

        if ((fp = fopen (DIRFILE, "wb")) == NULL)
                perror (DIRFILE),exit (1);

        for (g = root.groups; g < &root.groups[MAXUGR]; g++)
                *g = EMPTY;
        fwrite (&root, sizeof (DIR), 1, fp);            /* the root */
        treewrite (fp, "");                             /* the rest */
}


void treewrite (register FILE *fp, char *path)
{
        register DIR *p;

        for (p = dirlist (path); p != NULL; p = p->next)
        {       puts (p->name);
                fwrite (p, sizeof (DIR), 1, fp);
                treewrite (fp, p->name);
        }
}


DIR * dirlist (char *path)
{
        char searchpath [MAXNAMLEN];
        struct ffblk dirinfo;
        register struct ffblk *ip = &dirinfo;
        register DIR *p;
        DIR * list = NULL;
        u_int *g;

        strcpy (strpcpy (searchpath,path), "\\*.*");
        findfirst (searchpath, ip, FA_DIREC);

        do
        {
                if (!(ip->ff_attrib & FA_DIREC) ||  *ip->ff_name == '.')
                        continue;

                if ((p = (DIR *) calloc (1, sizeof (DIR))) == NULL)
                        perror ("dirlist"),exit (1);

                strcpy (strpcpy (p->name,searchpath) - 3, ip->ff_name);
                p->owner = 0;
                p->ispublic = 0;
                p->rights = R_NONE;
                for (g = p->groups; g < &p->groups[MAXUGR]; g++)
                        *g = EMPTY;
                p->next = list;
                list = p;
        }while (findnext (ip) == 0);
        return list;
}
```

```
/* freq.h - server request message structure and operation codes */

#ifndef FREQ_H
#define FREQ_H

#include "..\netdrv\udp.h"

#define MAXNAMLEN       40              /* maximum length of a path specification */

typedef struct                          /* request/reply message header structure */
{
        unsigned int seq;                       /* seqeunce no. of request */
        unsigned int id;                        /* user identification */
        char name [MAXNAMLEN];                  /* file/dir name */
        long pos;                               /* file pointer position */
        int code;                               /* operation code */
        int len;                                /* length of data field */
        unsigned int par1;                      /* extra parameter */
} FHDR;

#define FHDRLEN         sizeof (FHDR)
#define FMAXDAT         UMAXDAT - FHDRLEN

typedef struct                                  /* the complete request message */
{
        FHDR hdr;
        char data [FMAXDAT];
} FREQ;

#define FREQLEN         sizeof (FREQ)

/* some convenient aliases */
#define F_CNT   par1
#define F_MODE  par1
#define F_ATTR  par1
#define F_NBUF  pos

/* operation codes supported by server */
enum f_ops
{
        READ,
        WRITE,
        OPEN,
        UNLINK,
        RENAME,
        CHDIR,
        MKDIR,
        RMDIR,
        SEARCHF,
        SEARCHN,
        CREAT,
        CREATTEMP,
        CREATNEW,
        CHMOD,
        GETMOD,
```

```
        SETDT,
        GETDT,
        FLEN,
        CLOSE,
        GETSPACE,
        LOCK,
        UNLOCK,
        UTILS
};

/*      subfunctions for UTILS  */
enum log_ops
{       ARE_YOU_THERE = 0,                      /* check if rdr installed - supported locally */
        LOGSTAT,                                /* get log status - supported locally */
        LOGIN,
        LOGOUT
};

/* drive mapping - supported locally */
enum map_ops
{       GETMAP = LOGOUT + 1,
        SETMAP,
        DELMAP
};

#define GETUNO  (DELMAP + 1)

/* get/set information - supported on server */
enum info_ops
{       GETUSRINFO = GETUNO + 1,
        SETUSRINFO,
        GETGRPINFO,
        SETGRPINFO,
        GETDIRINFO,
        SETDIRINFO,
        GETUSRLIST,
        GETGRPLIST,
        GETUGRPS
};

/* user/group maintenance */
enum sys_ops
{       MKUSER = GETUGRPS + 1,
        RMUSER,
        MKGROUP,
        RMGROUP
};

#endif
```

```
/* str.h - str.asm function declarations */

char far *fstpcpy (char far *dest, char far *src);
char far *fmemcpy (char far *dest, char far *src, int n);
```

```
char *strdiff (char *s1, char *s2);
char far *strend (char far *str);
char *toascii2 (char *name, char far *fcbp);
char far *tofcb (struct fcb far *fcbp, char *name);
int pathcpy (char *dest, char far *src, char *limit);
```

```
/* rf.h - remote file state information structure */

#include "freq.h"

typedef struct
{
        char name [MAXNAMLEN];          /* pathname */
        int mode;                       /* access mode */
        int nhndls;                     /* no. of handles */
        long pos;                       /* file pointer position */
} L_FILE;

#define MAXF 20
#define MAXH 20
```

```
/* rmp.h - map function declarations */

char *pathmap (char *path, char far *p);
char *fcbmap (char *path, char far *fcbp);
void init_map (void);
```

```
/* rio.h - declarations for the server call interface */

#define NULL    0

extern FREQ pkt;
extern int errcode;

FREQ *servercall (FREQ *p, int code, int len);
int dev_open (void);
int dev_up (void);
void dev_down (void);
void selserver (long saddr);
```

```
/* rwhere.h - handler declarations */

#include "rf.h"

/* vector for original DOS roulines */
#define DOSVECT                 0x62

/* the handlers */
#define RCALL_HNDLRS            rspace,\
                                rdum,\
                                rdum,\
```

```
                                rmkdir,\
                                rrmdir,\
                                rchdir,\
                                rcreat,\
                                ropen,\
                                rclose,\
                                rread,\
                                rwrite,\
                                rdel,\
                                rseek,\
                                rmod,\
                                rioctl,\
                                rdum,\
                                rdum,\
                                rgetdir,\
                                rdum,\
                                rdum,\
                                rdum,\
                                rdum,\
                                rdum,\
                                rdum,\
                                rsearch,\
                                rsearch,\
                                rdum,\
                                rdum,\
                                rdum,\
                                rdum,\
                                rdum,\
                                rdum,\
                                rrename,\
                                rdl,\
                                rdum,\
                                rdum,\
                                rcreatemap,\
                                rcreat,\
                                rlock,\
                                rdum,\
                                rdum,\
                                rutils


#define digitize(c)             (((c) & 0x5f) - 'A')
#define local(drv)              ((drv) < n_drvs)
#define valid(drv)              ( (drv) < 26 && *map [(drv)] )

enum (  DOS = 0,                /* 4 possible paths for a request */
        FS,
        RET,
        RETERR );

extern char map [26][32];
extern L_FILE *fplab [];

void interrupt rint (REGPK reg,WORD ip,WORD cs,WORD flags);
int wherelo (REGPK  ss *regp);
```

```
int remotecall (REGPK _ss *regp);
int login (void);
```

```
;       rdr.asm - the redirector main routine
;       takes care of initialisation and installation

cr      equ     0dh
lf      equ     0ah

_TEXT   segment byte public 'CODE'
_TEXT   ends
_DATA   segment para public 'DATA'
_DATA   ends
_BSS    segment word public 'BSS'
bdata@  label byte
_BSS    ends
_BSSEND segment word public 'BSSEND'
_BSSEND ends

DGROUP  group   _TEXT,_DATA,_BSS,_BSSEND

ASSUME cs:_TEXT, ds:DGROUP

_DATA   segment

        msg1    db      'Redirector already installed',cr,lf,'$'
        msg2    db      'Driver not installed',cr,lf,'$'

_DATA   ends

_TEXT   segment

        ORG     0100H

rmain   proc    near

start:  mov     ax,cs
        mov     ds,ax
        call    near ptr bss_init

        call    near ptr chk_install            ;already installed?
        or      ax,ax
        jnz     rm0

        call    near ptr _chk_dev               ;driver installed?
        or      ax,ax
        jnz     rm1

        call    near ptr install

        mov     dx,offset DGROUP:endadr@
        add     dx,15
        mov     cx,4
        shr     dx,cl                           ;calculate size of memory to
```

```
        mov     ax,3100h                        ;KEEP
        int     21h

rm0:    mov     dx,offset DGROUP:msg1
        jmp     short rm2

rm1:    mov     dx,offset DGROUP: msg2
rm2:
        mov     ah,9
        int     21h                             ;print error message and
        mov     ax,4c01h                        ;abort
        int     21h

rmain   endp

;Initialize all variables in the _BSS segment to 0.
;Necessary because C functions assume that all uninitialized variables are at 0 by default.
;In Turbo C this is done by the start-up module c0l.obj, which is not included in the redirector.
;Adapted from the Turbo C start-up code c0.asm

bss_init        proc    near

        mov     ax,ds
        mov     es,ax
        xor     ax,ax
        mov     di,offset DGROUP:bdata@
        mov     cx,offset DGROUP:edata@
        sub     cx,di
        rep     stosb
        ret

bss_init        endp

_TEXT   ends

_BSSEND segment
edata@ label    byte
endadr@ label   byte
_BSSEND ends

extrn   chk_install : near
extrn   install : near
extrn   _chk_dev : near

end start
```

```
;       rint.asm - the int 21h handler for the redirector

__TINY__        equ     1

include rules.asm

_DOS    equ     0
_FS     equ     1
```

```
_RET    equ     2
_RETERR equ     3

pushall macro

        push    es
        push    ds
        push    bp
        push    di
        push    si
        push    dx
        push    cx
        push    bx
        push    ax

endm

popall  macro

        pop     a
        pop     bx
        pop     cx
        pop     dx
        pop     si
        pop     di
        pop     bp
        pop     ds
        pop     es

endm

Header@

CSeg@

_int    proc    far                     ;       the handler

        pushall                         ;       save all registers
        sti
        mov     bp,cs
        mov     ds,bp
        mov     bp,sp                   ;       bp points to registers on stack

        push    bp
        call    near ptr _whereto       ;       find out where this call is to be sent
        inc     sp
        inc     sp
        or      ax,ax
        jnz     h1

        popall
        int     62h                     ;       local call -> to DOS through int 62h
        jmp     short h5

h1:     cmp     al,_FS
```

```
        jne     h2

        push    bp
        call    near ptr _remotecall    ;       remote call -> to the server
        inc     sp
        inc     sp
        or      ax,ax
        jz      h4
        jmp     short h2

h2:     cmp     al,_RET                 ;       return to user
        je      h4

h3:     stc                             ;       return signalling error

h4:     popall

h5:     ret     2                       ;       throw away the original flags

_rint   endp

extrn   _whereip : near
extrn   _remotecall : near

CSegEnds

public  _rint

end
```

```
;       rmux.asm - the handler for the multiplex interrupt (int 2fh)

__TINY__        equ     1

include rules.asi

INTNO   equ     0C0h                    ;       the multiplex no.

pushall macro

        push    es
        push    ds
        push    bp
        push    di
        push    si
        push    dx
        push    cx
        push    bx
        push    ax

endm
```

```
popall  macro
        pop     ax
        pop     bx
        pop     cx
        pop     dx
        pop     si
        pop     di
        pop     bp
        pop     ds
        pop     es

endm

Header@

DSeg@

vec     dd      ?                       ; original int 2fh vector

DSegEnd@

CSeg@

raux            proc    far             ;       the handler

        cmp     ah,INTNO
        je      ri0                     ;       for me?
        jmp     cs:vec                  ;       chain to next handler

ri0:    or      al,al
        jne     ri1
        dec     al                      ;       al = 0? -> say I'm here by setting al = 0ffh
        iret

ri1:    pushall                         ;       save registers
        sti
        mov     bp,cs
        mov     ds,bp
        mov     bp,sp                   ;       bp points to  registers

        push    bp
        call    near ptr _rutils        ;       the utility handler
        inc     sp
        inc     sp
        or      ax,ax
        jz      ri2
        stc                             ;       carry flag reflects call status
ri2:
        popall
        ret     2

raux            endp
```

```
chk_install     proc    near            ;       check if rmux already installed

        mov     ah,INTNO
        xor     al,al
        int     2fh
        xor     ah,ah
        ret

chk_install     endp

install proc    near                    ;       install rmux in the chain of multiplex interrupt

        mov     al,2fh
        mov     ah,35h
        int     21h
        mov     word ptr vec,bx
        mov     word ptr vec + 2,es

        mov     ds,offset DGROUP:rmux
        mov     al,2fh
        mov     ah,25h
        int     21h
        ret

install endp

extern _rutils : near

CSegEnds

public chk_install
public install

end

/* rmp.c - drive mappings */

#include "asm.h"
#include "slr.h"

/* add a '\' if necessary */
#define slash(q)                if (*(q-1) != '\\')\
                                        *q++ = '\\'

extern BYTE n_drvs, cur_drv, req_drv;
extern char cur_dir [26][32];

char map [26][32];                      /* the map table */

/* build the absolute path specification from the path in p */
/* roughly, (path) = (map) + (current dir.) + (p) */
char *pathmap (char *path, char far *p)
{
        register int drv = req_drv;             /* drive specified in request */
        register char *q = path;
```

```
        char *limit;
        if ( *(p + 1) == ':' )
                p += 2;

        q = (char near *) fstpcpy (q, map [drv]);       /* copy out map */


        if (*p != '\\')
        {
                slash (q);
                q = (char near *) fstpcpy (limit = q, cur_dir [drv]);   /* current directory */
                slash (q);
                return ( (pathcpy (q, p, limit -1))? 0 : path);
        }
        else if (*(++p))
        {
                slash (q);
                return ( (pathcpy (q, p, q - 1))? 0 : path);
        }
        return path;
}

/* build absolute path specification from the filename in the FCB *fcbp */
char *fcbmap (char *path, char far *fcbp)
{
        register int drv = req_drv;
        register char *q = path;

        if ( *fcbp == -1 )              /* make sure that fcbp points to the filename */
                fcbp += 8;
        else if ( *fcbp <= 26 )
                fcbp++;

        q = (char near *) fstpcpy (q, map [drv]);               /* copy out map */
        slash (q);
        q = (char near *) fstpcpy (q, cur_dir [drv]);           /* current directory */
        slash (q);
        toasciiz (q, fcbp);                                     /* filename in ASCIIZ form */
        return path;
}

/* clear all mappings */
void init_map (void)
{
        register char (*m) [32] = map;

        while (m < &map[26])
                **m++ = '\0';
}
```

```
/* rio.c - driver interface */

#include "freq.h"
#include "str.h"
#include "..\netdrvr\dev.h"
#include <dos.h>
#include "..\fs\fserrno.h"

#define NULL    0

extern int echo;
extern unsigned uid;

FREQ pkt;                       /* the request message buffer */
int errcode;
static FREQ rbuf;
static struct cblk cblk = { 0L, 0, -1, 1, NULL, NULL };

FREQ *netio (FREQ *p, int len);

/* the server call interface */
FREQ *servercall (register FREQ *p, int code, int len)
{
        register FREQ *r;
        static unsigned seq;

        p->hdr.seq = ++seq;                             /* sequence no.of request */
        p->hdr.id = uid;                                /* user id */
        p->hdr.code = code;
        p->hdr.len = len;

        do {
                r = netio (p, FHDRLEN + len);           /* send request and wait for correct reply */
        } while (r->hdr.seq != seq);

        return ( ((errcode = r->hdr.code) == OK)? r : NULL);
}

/* driver interface */
static FREQ *netio (FREQ *p, int len)
{
        cblk.xbuf = p;                          /* pointer to request */

        _AX = 0x3d02;
        _DX = (unsigned) DEV;
        __int__ (0x62);

        _BX = _AX;
        _AX = 0x4401;
        _DX = 0xe0;
        __int__ (0x62);

        _AH = 0x40;
        _CX = len;
        _DX = (unsigned) &cblk;
```

```
        __int__ (0x62);                 /* send it and wait for reply */

        _AH = 0x3e;
        __int__ (0x62);
        return &rbuf;
}

/* check if driver installed */
int chk_dev (void)
{
        _AX = 0x3d02;
        _DX = (unsigned) DEV;
        __int__ (0x21);
        if (_FLAGS & 0x1)
                return 1;
        _BX = _AX;
        _AH = 0x3e;
        __int__ (0x21);
        return 0;
}

/* initialize driver */
int dev_up (void)
{
        register int hndl;
        cblk.rbuf = &rbuf;              /* pointer to buffer to hold replies */
        cblk.lport = -1;                /* acknowledgment expected */

        _AX = 0x3d02;
        _DX = (unsigned ) DEV;
        __int__ (0x21);
        hndl = _AX;
        if (_FLAGS & 0x1)
                return 1;

        _BX = hndl;
        _AX = 0x4402;
        _DX = (unsigned) &cblk;         /* open port */
        __int__ (0x21);
        if (cblk.lport == -1)
                return 1;

        _BX = hndl;
        _AH = 0x3e;
        __int__ (0x21);
        return 0;
}

/* shut down driver */
void dev_down (void)
{
        register int hndl;

        _AX = 0x3d02;
        _DX = (unsigned) DEV;
```

```
        __int__ (0x21);
        _BX = hndl = _AX;
        _AX = 0x4403;                           /* close port */
        _DX = (unsigned) &cblk;
        __int__ (0x21);

        _BX = hndl;
        _AH = 0x3e;
        __int__ (0x21);
}


void setserver (long saddr)
{
        cblk.fhost = saddr;                     /* set server internet address */
}
```

```
/* rwhere.c - decide where to send a request */

#include "asm.h"
#include "rwhere.h"
#include "int21.h"

extern BYTE cur_drv, n_drvs, log;
BYTE req_drv;
char far *dta;


int whereto (register REGPK _ss * r)
{
        register int drv;
        char far *p;

        switch (r->FUNC)                        /* decision based on function no. */
        {
                case 0x39 :
                case 0x3a :                     /* ASCIIZ calls */
                case 0x3b :                     /* check drive letter specified in path */
                case 0x3c :                     /* or default drive */
                case 0x3d :
                case 0x41 :
                case 0x43 :
                case 0x4b :
                case 0x4e :
                case 0x56 :
                case 0x5a :
                case 0x5b :     drv = (*(BUFFER + 1) == ':')? digitize (*BUFFER) : cur_drv;
                                break;

                case 0x4f :     drv = (drv = *dta)? drv - 1 : cur_drv;
                                break;

                case 0x3e :
                case 0x3f :
```

```
        case 0x40 :                     /* handle - based calls */
        case 0x42 :                     /* check if handle is remote */
        case 0x44 :
        case 0x45 :
        case 0x46 :
        case 0x57 :
        case 0x5c :     return ( (r->HNDL < MAXH && fptab [r->HNDL])? FS : DOS );

        /* set drive - intercepted to keep track of changes */
        case 0x0e :     return ( local ((cur_drv = r->DRVCODE))? DOS
                                : (r->b.al = n_drvs, RET));

        /* get drive */
        case 0x19 :     r->b.al = cur_drv;
                        return RET;

        /* note changes in DTA */
        case 0x1a :     dta = BUFFER;
                        return DOS;

        /* FCB search, delete, rename calls */
        case 0x11 :
        case 0x12 :
        case 0x13 :
        case 0x17 :     if ( *(p = BUFFER) == -1)
                                p += 7;

                        drv = (*p)? *p - 1  :  cur_drv;
                        return ( local (drv)? DOS
                                : valid (drv)? (req_drv = drv, FS)
                                : (r->b.al = 0xff, RETERR));

        /* parse filename */
        case 0x29 :     if (*(FCP (r->w.ds, r->w.si) + 1) != ':')
                                return DOS;

                        drv = digitize ( *FCP (r->w.ds, r->w.si));
                        return ( local (drv)? DOS
                                : !valid (drv)? (r->b.al = 0xff, RETERR)
                                : (r->w.si += 2, r->b.al != 0x2,
                                   *FCP (r->w.es, r->w.di) = drv + 1,
                                   DOS));

        /* disk space, get current directory */
        case 0x1c :
        case 0x36 :
        case 0x47 :     drv = (r->DRVCODE)? (r->DRVCODE - 1) : cur_drv;
                        break;

        default :       return DOS;
    }
    return ( local (drv)? DOS                   /* route the request based on the drive */
        :valid (drv)? (req_drv = drv, FS)
        :(r->ERRCODE = 3,RETERR));
}
```

```
/* demultiplex request to the various handlers */
int remotecall (register REGPK _ss *r)
{
        register int code;
        static int (*rcall_hndlr []) (REGPK _ss *r) =
        {
                RCALL_HNDLRS
        };

        switch (r->FUNC)
        {
                case 0x11 :
                case 0x12 :     r->b.al = (code = rfcbsearch (r))? 0xff : 0;
                                break;

                case 0x13 :     r->b.al = (code = rdel (r))? 0xff : 0;
                                break;

                case 0x17 :     r->b.al = (code = rfcbrename (r))? 0xff : 0;
                                break;

                case 0x1c :     if (code = rspace (r))
                                        r->ERRCODE = code;
                                break;

                default :       if (code = (*rcall_hndlr [r->FUNC - FIRST]) (r))
                                        r->ERRCODE = code;
                                break;
        }
        return ( code? 1 : 0);
}
```

```
/* rmrq.c - miscellaneous requests */

#pragma inline

#include <dos.h>
#include <errno.h>
#include "asm.h"
#include "int21.h"
#include "freq.h"
#include "str.h"
#include "..\fs\fac.h"
#include "..\fs\fserrno.h"
#include "rio.h"
#include "rmp.h"

#define NUTILS  ( sizeof (util_hndlr) / sizeof (int (*) ()) )
#define DOSVECT 0x62

int rstat (REGPK _ss *r);
int _login (REGPK _ss *r);
int _logout (REGPK _ss *r);
```

```
int rmap (REGPK _ss *r);
int rdum ();
int rinfo (REGPK _ss *r);
int runo (REGPK _ss *r);
void interrupt rint (REGPK reg,WORD ip,WORD cs,WORD flags);

unsigned uid;                           /* user identification */
unsigned uno;                           /* user number */

BYTE n_drvs, cur_drv;                   /* no. of logical drives, current drive */
BYTE log = 0;                           /* whether logged in or not */
extern char map [26][32];
extern char cur_dir [26][32];

        static int (*util_hndlr []) (REGPK _ss *) =             /* utility handlers */
        {
                rdum,
                rstat,
                _login,
                _logout,
                rmap,
                rmap,
                rmap,
                runo,
                rinfo,
                rinfo,
                rinfo,
                rinfo,
                rinfo,
                rinfo,
                rinfo,
                rinfo,
                rinfo,
                rinfo,
                rinfo,
                rinfo,
                rinfo,
                rinfo
        };


int rlogin (unsigned userno, char far *pwd)
{
        register FREQ *p = &pkt;

        fstpcpy (p->hdr.name, pwd);
        p->hdr.par1 = uno = userno;             /* save user no. */

        if ((p = servercall (p, UTILS + LOGIN, 0)) == NULL)
                return errcode;

        uid = p->hdr.par1;                      /* save user id */
        return p->hdr.code;
}
```

```
int rlogoff (void)
{
        *pkt.hdr.name = '\0';
        return ((servercall (&pkt, UTILS + LOGOUT, 0) == NULL)? errcode : OK);
}


int rdum ()
{
        return EINVFNC;
}

/* 1ch, 36h - disk space */
int rspace (register REGPK _ss *r)
{
        register FREQ *p = &pkt;

        *p->hdr.name = '\0';
        if ((p = servercall (p, GETSPACE, 0)) == NULL)
                return 0xffff;

        if (r->FUNC == 0x36)
                r->w.bx = ((struct dfree *) p->data)->df_avail;
        else
                *FCP (r->w.ds, r->w.bx) = 0xf8;
        r->w.ax = ((struct dfree *) p->data)->df_sclus;
        r->w.cx = ((struct dfree *) p->data)->df_bsec;
        r->w.dx = ((struct dfree *) p->data)->df_total;
        return OK;
}

/* the utility handlers */
int rutils (REGPK _ss *r)
{
        register int code;

        return ( (r->SUBFUNC >= NUTILS)? (r->ERRCODE = EINVFNC)
                 : (code = (*util_hndlr [r->SUBFUNC]) (r)) != OK ? (r->ERRCODE = code)
                 : OK );

}

/* drive mappings */
int rmap (REGPK _ss *r)
{
        register FREQ *p = &pkt;
        register int drv = r->b.bl;

        if (drv < n_drvs || drv > 25)           /* drive valid? */
                return EDRV;
        switch (r->SUBFUNC)
        {
                case SETMAP :   fstpcpy (p->hdr.name, BUFFER);
                                if ((p = servercall (p, CHDIR, 0)) == NULL)          /* check if path OK */
                                        return errcode;
```

```
                        fstpcpy (map [drv], p->hdr.name);          /* save map */
                        *cur_dir [drv] = '\0';
                        break;

                case GETMAP :   fstpcpy (BUFFER, map [drv]);
                        break;

                case DELMAP :   if (drv == cur_drv)
                                        return EDRV;
                        *map [drv] = '\0';                         /* reset map */
                        break;

                default :       return EINVFNC;
        }
        return OK;
}

/* user/group/directory information handlers */
int rinfo (register REGPK _ss *r)
{
        register FREQ *p = &pkt;
        register int len;

        *p->hdr.name = len = 0;
        switch (r->SUBFUNC)
        {
                case SETUSRINFO :       fmemcpy (p->data, BUFFER, len = sizeof (USER));
                case GETUSRINFO :
                case GETUGRPS   :
                case RMUSER     :
                case RMGROUP    :       p->hdr.par1 = r->w.cx;
                                        break;

                case SETGRPINFO :       fmemcpy (p->data, BUFFER , len = sizeof (GROUP));
                case GETGRPINFO :       p->hdr.par1 = r->w.cx;
                                        break;

                case SETDIRINFO :
                case GETDIRINFO :       fmemcpy (p->data, BUFFER, len = sizeof (DIR));
                                        fstpcpy (p->hdr.name, BUFFER);
                                        break;

                case MKUSER     :
                case MKGROUP    :       fstpcpy (p->data, BUFFER);
                                        len = MAXUNAM;
                                        break;
        }

        if ((p = servercall (p, UTILS + r->SUBFUNC, len)) == NULL)
                return errcode;
```

```
        switch (r->SUBFUNC)
        {
                case GETUSRINFO :
                case GETGRPINFO :
                case GETDIRINFO :
                case GETUSRLIST :
                case GETGRPLIST :
                case GETUGRPS   :       fmemcpy (BUFFER, p->data, p->hdr.len);
                                        break;

                case MKUSER     :
                case MKGROUP    :       r->w.cx = p->hdr.par1;
                                        break;

        }
        return OK;
}

/* get user no. */
int runo (REGPK _ss *r)
{
        r->w.ax = uno;
        return OK;
}

/* remote login */
int _login (REGPK _ss *r)
{
        int code;

        if (log)
                return ELOGGED;                 /* already logged in */
        if (dev_up () != 0)                     /* unable to open driver port */
                return EPORT;

        /* set int 62h to point to DOS */
        asm     mov     ah,35h
        asm     mov     al,21h
        asm     int     21h
        asm     mov     ax,es
        asm     mov     ds,ax
        asm     mov     dx,bx
        asm     mov     ah,25h
        asm     mov     al,DOSVECT
        asm     int     21h
        asm     mov     ax,cs
        asm     mov     ds,ax

        setserver (MK_LONG (r->w.si, r->w.di));

        if ((code = rlogin (r->w.bx, BUFFER)) != OK)            /* attempt login */
        {       dev_down ();
                return code;
        }
```

```
        log = 1;                                        /* success */

        asm     mov     ah,19h
        asm     int     21h
        asm     mov     cur_drv,al                      /* initialize current drive */
        asm     mov     ah,0eh
        asm     mov     dl,al
        asm     int     21h
        asm     mov     n_drvs,al                       /* no. of logical drives */

        asm     mov     ah,25h
        asm     mov     al,21h
        asm     mov     dx,offset DGROUP:rint
        asm     int     21h                             /* 'switch on' redirector */

        init_map ();                                    /* reset all drive mappings */
        return OK;
}


int _logout (REGPK _ss *r)
{
        int code;

        if (!log)
                return ENLOGGED;
        if ((code = rlogoff ()) != OK)
                return code;

        log  = 0;
        /* restore interrupt vectors */
        asm     mov     ah,35h
        asm     mov     al,DOSVECT
        asm     int     DOSVECT
        asm     mov     ax,es
        asm     mov     ds,ax
        asm     mov     dx,bx
        asm     mov     ah,25h
        asm     mov     al,21h
        asm     int     DOSVECT
        asm     mov     ax,cs
        asm     mov     ds,ax

        dev_down ();                            /* close driver port */
        return OK;
}

/* LOGSTAT - return log status */
int rstat (REGPK _ss *r)
{
        r->w.ax = log;
        return OK;
}
```

```
/* rdrq.c - remote directory requests */

#include <dos.h>
#include <errno.h>
#include "asm.h"
#include "int21.h"
#include "freq.h"
#include "str.h"
#include "rio.h"
#include "rmp.h"
#include "..\fs\fserrno.h"

extern BYTE req_drv;
extern char map [26][32];

char cur_dir [26][32];                  /* the current directory table */


/* 39h - MKDIR */
int rmkdir (REGPK _ss *r)
{
        register FREQ *p = &pkt;

        if (pathmap (p->hdr.name, BUFFER) == 0)
                return ENOPATH;

        return ( (servercall (p, MKDIR, 0) == NULL)? errcode : OK );
}

/* 3ah - RMDIR */
int rrmdir (REGPK _ss *r)
{
        register FREQ *p = &pkt;

        if (pathmap (p->hdr.name, BUFFER) == 0)
                return ENOPATH;

        return ( (servercall (p, RMDIR, 0) == NULL)? errcode : OK );
}

/* 3bh - CHDIR */
int rchdir (REGPK _ss *r)
{
        register char *q;
        register FREQ *p = &pkt;

        if (pathmap (p->hdr.name, BUFFER) == 0)
                return ENOPATH;
        if ((p = servercall (p, CHDIR, 0)) == NULL)             /* check if change is valid */
                return errcode;
        if ( *(q = strdiff (p->hdr.name, map [req_drv])) == '\\')
                q++;
        fstpcpy (cur_dir [req_drv], q);                 /* update current directory table */
        return OK;
}
```

```
/* function 47h - get current directory : handled locally */
int rgetdir (register REGPK _ss *r)
{
        fstpcpy (FCP (r->BUFSEG, r->w.si), cur_dir [req_drv]);          /* copy out from table */
        return OK;
}
```

```
/* rfrq.c - remote file requests */

#include <dos.h>
#include <errno.h>
#include "asm.h"
#include "int21.h"
#include "freq.h"
#include "str.h"
#include "rf.h"
#include "rio.h"
#include "rmp.h"
#include "..\fs\fserrno.h"

#define min(a,b)        ( ( (a) < (b) )? (a) : (b) )
#define digitize(c)     (((c) & 0x5f) - 'A')

extern BYTE req_drv, cur_drv;

L_FILE *fptab [MAXH];                           /* remote file state maintenance table */

int opendev (char *name, int mode);
int lopen (char *path, int mode);
int lclose (int hndl);
L_FILE *falloc (void);
int halloc (void);

/* 3dh - OPEN */
int ropen (register REGPK _ss *r)
{
        int code;
        register FREQ *p = &pkt;
        static char name [MAXNAMLEN] = "?:";

        fstpcpy (name + 2, BUFFER);
        if ((code = opendev (name, r->MODE)) < 0)               /* make sure it is not a local device */
        {
                if (pathmap (p->hdr.name, BUFFER) == 0)
                        return ENOPATH;

                p->hdr.F_MODE = r->MODE;
                if ((p = servercall (p, OPEN, 0)) == NULL)              /* remote open */
                        return errcode;

                if ( (code = lopen (p->hdr.name, r->MODE)) == -1)       /* local open */
                        return EMFILE;
        }
```

```
        r->RHNDL = code;
        return OK;
}

/* check if file is a device, in which case it must be opened locally */
int opendev (char *name, int mode)
{
        _AH = 0x3d;
        _AL = mode;
        _DX = (unsigned) name;
        geninterrupt (0x62);
        return ( (_FLAGS & 0x1)? -1 : _AX );
}

/* keep local records about the file for later reference */
int lopen (char *path, int mode)
{
        register int hndl;
        register L_FILE *f;

        /* allocate an L_FILE structure to store state and get a handle from DOS */
        if ( (f = falloc ()) == NULL || (hndl = halloc ()) == -1)
                return -1;

        fstpcpy (f->name, path);                /* save state info */
        f->mode = mode;
        f->pos = 0L;
        fptab [hndl] = f;                       /* keep track of L_FILE through the handle */
        return hndl;
}

/* allocate space to store state info for remote files */
static L_FILE *falloc (void)
{
        static L_FILE ftab [MAXF];
        register L_FILE *f;

        for (f = ftab; f < &ftab [MAXF]; f++)
                if (f->nhndls == 0)                     /* find an unused entry */
                        return (f->nhndls = 1, f);

        return NULL;
}

/* get a handle for the file : from DOS to avoid clashes */
static int halloc (void)
{
        _AH = 0x45;
        _BX = 2;                                /* DUP the stderr handle */
        geninterrupt (0x62);
        return ( (_FLAGS & 0x1)? -1 : _AX);
}
```

```
/* 3eh - CLOSE */
int rclose (register REGPK _ss *r)
{
        register FREQ *p = &pkt;

        fstpcpy (p->hdr.name, fptab [r->HNDL]->name);          /* get name */
        servercall (p, CLOSE, 0);                                    /* remote close */
        return ( lclose (r->HNDL) );                           /* local close */
}

/* 3fh - READ */
int rread (REGPK _ss *r)
{
        register FREQ *p = &pkt;
        L_FILE *f = fptab [r->HNDL];
        register unsigned i = r->LEN;
        unsigned n;
        char far *buf = BUFFER;

        if ( f->mode & 0x1 )                                   /* access mode checked locally */
                return EACCES;

        fstpcpy (p->hdr.name, f->name);                        /* retreive state */
        p->hdr.pos = f->pos;

        do {
                p->hdr.F_CNT = n = min (i, FMAXDAT);

                if ((p = servercall (p, READ, 0)) == NULL)
                        return errcode;

                fmemcpy (buf, p->data, p->hdr.F_CNT);
                i -= p->hdr.F_CNT;                             /* update count and buffer position */
                buf += p->hdr.F_CNT;
        } while (p->hdr.F_CNT == n  &&  i > 0);

        f->pos = p->hdr.pos;                                   /* save state */
        r->CNT = r->LEN - i;
        return OK;
}

/* 40h - WRITE */
int rwrite (REGPK _ss *r)
{
        register FREQ *p = &pkt;
        L_FILE *f = fptab [r->HNDL];
        register unsigned i = r->LEN;
        unsigned n;
        char far *buf = BUFFER;

        if ( (f->mode & 0x3) == 0 )
                return EACCES;

        fstpcpy (p->hdr.name, f->name);
        p->hdr.pos = f->pos;
```

```
        do {
                p->hdr.F_CNT = n = min (i, FMAXDAT);

                fmemcpy (p->data, buf, p->hdr.F_CNT);

                if ((p = servercall (p, WRITE, p->hdr.F_CNT)) == NULL)
                        return errcode;

                i -= p->hdr.F_CNT;
                buf += p->hdr.F_CNT;
        } while (p->hdr.F_CNT == n  &&  i > 0);

        f->pos = p->hdr.pos;
        r->CNT = r->LEN - i;
        return OK;
}


/* 41h - DELETE */
int rdel (REGPK _ss *r)
{
        register FREQ *p = &pkt;

        if (r->FUNC == 0x13)
                fcbmap (p->hdr.name, BUFFER);
        else
                if (pathmap (p->hdr.name, BUFFER) == 0)
                        return ENOPATH;

        return ( (servercall (p, UNLINK, 0) == NULL)? errcode : OK );
}

/* 56h - RENAME */
int rrename (register REGPK _ss *r)
{
        int drv;
        register FREQ *p = &pkt;

        drv = ( *(FCP (r->w.es, r->w.di) + 1) == ':' )?
                        digitize ( *FCP (r->w.es, r->w.di) ) : cur_drv;
        if (drv != req_drv)
                return ENOTSAM;                 /* not same drive */

        if (pathmap (p->hdr.name, BUFFER) == 0  ||
            pathmap (p->data, FCP (r->w.es, r->w.di)) == 0)
                return ENOPATH;
        return ((servercall (p, RENAME, MAXNAMLEN) == NULL)? errcode : OK);
}


int rfcbrename (REGPK _ss *r)
{
        register FREQ *p = &pkt;

        fcbmap (p->hdr.name, BUFFER);
```

```
        fcbmap (p->data, BUFFER + 17);
        return ( (servercall (p, RENAME, MAXNAMLEN) == NULL)? errcode : OK);
}


int rioctl (REGPK _ss *r)
{
        switch (r->b.al)
        {
                case 0x0 :      r->w.dx = req_drv;
                                return OK;

                case 0x6 :
                case 0x7 :      r->b.al = 0xff;         /* always ready */
                                return OK;

                default :       return EINVFNC;
        }
}


/* CREAT, CREATNEW */
int rcreat (register REGPK _ss *r)
{
        int code;
        register FREQ *p = &pkt;

        p->hdr.F_ATTR = r->ATTR;
        if (pathmap (p->hdr.name, BUFFER) == 0)
                return ENOPATH;

        if ((p = servercall (p, (r->FUNC == 0x3c)? CREAT : CREATNEW, 0)) == NULL)
                return errcode;

        if ( (code = lopen (p->hdr.name, 2)) == -1)             /* open locally */
                return EMFILE;
        r->RHNDL = code;
        return OK;
}

/* CREATTEMP */
int rcreattemp (REGPK _ss *r)
{
        register FREQ *p = &pkt;
        int code;
        char *tail;

        p->hdr.F_ATTR = r->ATTR;
        if (pathmap (p->hdr.name, BUFFER) == 0)
                return ENOPATH;

        tail = (char near *) strend (p->hdr.name);

        if ((p = servercall (p, CREATTEMP, 0)) == NULL)
                return errcode;
```

```
        if ( (code = lopen (p->hdr.name,-2)) == -1)
                return EMFILE;
        r->RHNDL = code;
        fstpcpy ( strend (BUFFER), tail);              /* return complete name */
        return OK;
}

/* CHMOD, GETMOD */
int rmod (register REGPK _ss *r)
{

        register FREQ *p = &pkt;

        if (pathmap (p->hdr.name, BUFFER) == 0)
                return ENOPATH;
        p->hdr.F_ATTR = r->ATTR;

        if ((p = servercall (p, (r->SUBFUNC == 0x1)? CHMOD : GETMOD, 0)) == NULL)
                return errcode;

        r->ATTR = p->hdr.F_ATTR;
        return OK;
}

/* DATE, TIME */
int rdt (register REGPK _ss *r)
{
        register FREQ *p = &pkt;

        fstpcpy (p->hdr.name, fptab [r->HNDL]->name);
        *( (unsigned *) p->data ) = r->TIME;
        *( (unsigned *) p->data + 1 ) = r->DATE;

        if ((p = servercall (p, (r->SUBFUNC == 0x1)? SETDT : GETDT, 2 * sizeof (int))) == NULL)
                return errcode;

        r->TIME = *( (unsigned *) p->data);
        r->DATE = *( (unsigned *) p->data + 1);
        return OK;
}

/* 42h - SEEK */
int rseek (register REGPK _ss *r)
{
        FREQ *p = &pkt;
        register long *posp = &fptab [r->HNDL]->pos;

        switch (r->b.al )                                   /* position specified from: */
        {
                case 0 :        *posp =  OFFSET;            /* start of file */
                                break;

                case 1 :        *posp += OFFSET;            /* current position */
                                break;
```

```
                case 2 :        fstpcpy (p->hdr.name, fptab [r->HNDL]->name);   /* end of file */
                                if ((p = servercall (p, FLEN, 0)) == NULL)              /* get file length */
                                        return errcode;
                                *posp = p->hdr.pos + OFFSET;
                                break;

                default :       return EINVFNC;
        }
        r->w.dx = (unsigned)((unsigned long) *posp >> 16);
        r->w.ax = (unsigned) *posp;
        return OK;
}

/* 5ch - LOCK */
int rlock (REGPK _ss *r)
{
        FREQ *p = &pkt;

        fstpcpy (p->hdr.name, fptab [r->HNDL]->name);
        p->hdr.pos = MK_LONG (r->w.cx, r->w.dx);
        *(long *) p->data = MK_LONG (r->w.si, r->w.di);
        return ( (servercall (p, (r->SUBFUNC)? UNLOCK : LOCK, sizeof (long)) == NULL)?
                        errcode : OK);
}

/* 45h - DUP : local */
int rdup (REGPK _ss *r)
{
        register int hndl;

        if ( (hndl = halloc ()) == -1)          /*allocate a handle */
                return EMFILE;
        ( fptab [hndl] = fptab [r->HNDL] )->nhndls++;
        return OK;

}

/* 46h - CDUP : local */
int rcdup (REGPK _ss *r)
{
        if (r->w.cx >= MAXH)
                return EMFILE;
        lclose (r->w.cx);
        ( fptab [r->w.cx] = fptab [r->HNDL] )->nhndls++;
        return OK;
}

/* close locally */
int lclose (register int hndl)
{
        if ( hndl < MAXH  &&  fptab [hndl] != NULL )
        {
                fptab [hndl]->nhndls--;
                fptab [hndl] = NULL;
        }
```

```
        _AH = 0x3e;
        _BX = hndl;
        geninterrupt (0x62);
        return ( (_FLAGS & 0x1)? EBADF : OK);
}
```

```
/* rsrq.c - remote search requests */

#include <dos.h>
#include <dir.h>
#include <errno.h>
#include "asm.h"
#include "int21.h"
#include "freq.h"
#include "str.h"
#include "rio.h"
#include "rmp.h"
#include "..\fs\fserrno.h"

#define INFOSIZE        sizeof (struct ffblk)

/* no. of buffers to hold search info : depends on amount of data that can be sent in one go */
#define NBUFS           ((FMAXDAT / INFOSIZE) - 1)

extern BYTE req_drv;
extern char far *dta;

static struct ffblk sbuf [NBUFS];               /* the search info buffer */
static struct ffblk *s;                         /* pointer to info in buffer */
static int n;                                   /* no. of matching files remaining */

void xlate (struct ffblk *s, int is_xfcb);

struct _fcb                             /* unopened FCB returned by functions 11h,12h */
{
        char drive;
        char name[8];
        char ext[3];
        char attr;
        char resvd[10];
        unsigned time;
        unsigned date;
        unsigned scn;
        long filsize;
};

struct _xfcb                            /* unopened extended FCB */
{
        char flag;
        char resvd[5];
        char attr;
        struct _fcb fcb;
};
```

```
/* 4eh, 4fh - search with ASCIIZ paths */
int rsearch (REGPK _ss *r)
{
        register FREQ *p = &pkt;

        switch (r->FUNC)
        {
                case 0x4e :     p->hdr.F_ATTR = r->ATTR;
                                p->hdr.F_NBUF = NBUFS + 1;
                                if (pathmap (p->hdr.name, BUFFER) == 0)
                                        return ENOPATH;

                                if ((p = servercall (p, SEARCHF, 0)) == NULL)          /* first match(es) */
                                        return errcode;
                                break;

                case 0x4f :     if (n > 0)                      /* if buffer not empty, copy out from it */
                                {
                                        fmemcpy (dta, (char far *) s++, INFOSIZE);
                                        *dta = req_drv + 1;
                                        n--;
                                        return OK;
                                }

                                if (s < &sbuf [NBUFS])          /* no more files */
                                        return ENOFILE;

                                p->hdr.F_NBUF = NBUFS + 1;
                                *p->hdr.name = '\0';
                                fmemcpy ( p->data, (char far *) &sbuf [NBUFS - 1], INFOSIZE);
                                if ((p = servercall (p, SEARCHN, INFOSIZE)) == NULL)           /*get next matches */
                                        return errcode;
                                break;

                default :       return EINVFNC;
        }

        fmemcpy (dta, p->data, INFOSIZE);               /* first info copied out to user */
        *dta = req_drv + 1;
        /* the rest goes into the buffer */
        s = (struct ffblk near *)
            fmemcpy ( (char far *) sbuf, (char far *) ((struct ffblk *) p->data + 1),
                                        (n = p->hdr.F_NBUF - 1) * INFOSIZE );
        return OK;
}

/* 11h,12h */
int rfcbsearch (REGPK _ss *r)
{
        register FREQ *p = &pkt;
        register int is_xfcb = ( *BUFFER == -1 );       /*extended FCB specified? */
```

```
        switch (r->FUNC)
        {
                case 0x11 :     p->hdr.F_ATTR = (is_xfcb)? *(BUFFER + 6) : 0;
                                p->hdr.F_NBUF = NBUFS + 1;
                                fcbmap (p->hdr.name, BUFFER);

                                if ((p = servercall (p, SEARCHF, 0)) == NULL)
                                        return 1;
                                break;

                case 0x12 :     if (n > 0)
                                {
                                        xlate (s++, is_xfcb);           /* copy out to user */
                                        n--;
                                        return OK;
                                }

                                if (s < &sbuf [NBUFS])
                                        return ENOFILE;

                                p->hdr.F_NBUF = NBUFS + 1;
                                *p->hdr.name = '\0';
                                fmemcpy (p->data, (char far *) &sbuf [NBUFS - 1], INFOSIZE);
                                if ((p = servercall (p, SEARCHN, INFOSIZE)) == NULL)
                                        return 1;
                                break;

                default :       return EINVFNC;
        }

        xlate ( (struct ffblk *) p->data, is_xfcb);
        s = (struct ffblk near *)
            fmemcpy ( (char far *) sbuf, (char far *) ((struct ffblk *) p->data + 1),
                                        (n = p->hdr.F_NBUF - 1) * INFOSIZE );
        return OK;
}

/* parse search info into the FCB at the current DTA */
void xlate (register struct ffblk *s, int is_xfcb)
{
        struct _fcb far *f = (struct _fcb far *) dta;

        if (is_xfcb)
        {
                ( (struct _xfcb far *) f)->flag = -1;
                ( (struct _xfcb far *) f)->attr = s->ff_attrib;
                f = &( ( (struct _xfcb far *) f)->fcb);
        }
        tofcb ((struct fcb far *)f, s->ff_name);
        f->filsize = s->ff_fsize;
        f->date = s->ff_fdate;
        f->time = s->ff_ftime;
        f->attr = s->ff_attrib;
        f->drive = req_drv + 1;
}
```

```
/* pclan.h - PC-LAN frame declarations */

#ifndef PCLAN_H
#define PCLAN_H

#define         MAXPKTLEN       255

struct pkthdr                           /* the packet header */
{       unsigned char dest;
        unsigned char ctrl;
        unsigned char src;
        unsigned char type;
        unsigned int len;
};

#define         PHDRLEN         sizeof (struct pkthdr)
#define         PMAXDAT         (MAXPKTLEN - PHDRLEN)

typedef struct
{
        struct pkthdr hdr;
        unsigned char data [PMAXDAT];
} PACKET;

#define         IP              9
#define         MYBCAST         0xff
#define         MYGATEWAY       0

#endif
```

```
/* buf.h - declarations for buffer management */

#ifndef BUF_H
#define BUF_H

#include "pclan.h"
#include <stddef.h>

typedef struct buffer
{       PACKET pkt;             /* the actual buffer */
        int     status;         /* whether the buffer is free or reserved */
        struct buffer *next;
} BUFFER;


#define NBUFS   4

#define B_FREE  0
#define B_RSVD  1

PACKET *allocp (void);
void freep (PACKET *p);

#endif
```

```
/* q.h - declarations for queue management */

#ifndef Q_H
#define Q_H

#include "pclan.h"
#include "buf.h"
#include <stddef.h>

typedef struct
{       BUFFER *head;           /* pointer to the packet at the head of the queue */
        BUFFER *tail;           /* pointer to the one at the tail */
} Q;

#define isempty(q)      ((q)->head == NULL)

void enque (Q *q, PACKET *p);
PACKET *deque (Q *q);

#endif
```

```
/* port.h - declarations for port management */

#ifndef PORT_H
#define PORT_H

#include "pclan.h"
#include "q.h"

typedef struct
{       Q q;                    /* the port queue */
        int status;             /* status of this port : free or reserved */
        int dummy;              /* just to round off the size to 8 bytes (makes random access easier) */
} PORT;

#define NPORTS  4

#define P_FREE          0
#define P_RSVD          1

#define NTRIES          0x40000L        /* no.of retries for precv */
#define ANYPORT         0xffff

#define pvalid(n)       ((n) < NPORTS  &&  portE(n)].status == P_RSVD)

int psend (unsigned pno, PACKET *p);
PACKET *precv (unsigned pno);
int pstat (unsigned pno);
int popen (unsigned pno);
void pclose (unsigned pno);
void pclear (unsigned pno);

#endif
```

```
/* udp.h - declarations for UDP */

#ifndef UDP_H
#define UDP_H

#include "ip.h"

struct udphdr
{       unsigned int sport;
        unsigned int dport;
        unsigned int len;
        unsigned int chksum;
};

#define         UHDRLEN         sizeof (struct udphdr)
#define         UMAXDAT         (IMAXDAT - UHDRLEN)

struct udp
{       struct udphdr hdr;
        unsigned char data [UMAXDAT];
};

typedef struct
{       IPADDR  fhost;
        unsigned int fport;
        unsigned int lport;
} UDPCONN;

struct phdr                             /* the UDP pseudo-header */
{       IPADDR  src;
        IPADDR  dest;
        unsigned char zero;
        unsigned char prot;
        unsigned len;
};

#define         UDPPTR(p)       ((struct udp *) IPPTR (p)->data)

int udpsend (UDPCONN *u, PACKET *p, int datlen);
int udpdemux (PACKET *p, int len);

#endif
```

```
/* ip.h - declarations for IP */

#ifndef IP_H
#define IP_H

#include "pclan.h"

typedef long IPADDR;            /* internet address */
```

```
struct iphdr
{       unsigned char ver_hlen;
        unsigned char srvctyp;
        unsigned int len;
        unsigned int id;

          unsigned int frag;
          unsigned char ttl;
          unsigned char prot;
          unsigned int chksum;
          IPADDR  src;
          IPADDR  dest;
 };

 #define        IHDRLEN         sizeof (struct iphdr)
 #define        IMAXDAT         (PMAXDAT - IHDRLEN)

 struct ip
 {      struct iphdr hdr;
        unsigned char data [IMAXDAT];
 };

 #define        IVER_HLEN       0x54            /* byte swapped */
 #define        ISRVCTYP        0
 #define        IFRAG           0
 #define        ITTL            0xff

 #define        UDP_PROT        17

 #define        IPPTR(p)        ((struct ip *) p->data)
 #define        getnet(addr)    ((addr) & 0xfff0)
 #define        getnode(addr)   ((unsigned char) (addr))

 int ipsend (IPADDR fhost, PACKET *p, int datlen);
 int ipdemux (PACKET *p, int len);

 #endif
```

```
/* dev.h - declarations for the network driver interface */

struct cblk                     /* the device control block */
{       long fhost;             /* foreign internet address */
        int fport;              /* foreign port no. */
        int lport;              /* local port no. */
        int ack;                /* acknowledgment expected */
        void far *xbuf;         /* data to be sent */
        void far *rbuf;         /* buffer to hold received data */
};


#define DEV     "NET"           /* the device driver name */
```

```
;pclan.inc - PC-LAN frame declarations

MAXPKTLEN       equ     255

phdr    struc                           ;       packet header
        dest    db      ?
        ctrl    db      ?
        src     db      ?
        type    db      ?
        len     dw      ?
phdr    ends

iphdr   struc
                db      12 dup (?)
        ip_src  dd      ?
                dd      ?
iphdr   ends

udphdr  struc
        udp_sport       dw      ?
        udp_dport       dw      ?
        udp_len         dw      ?
                        dw      ?
udphdr  ends

PHDRLEN         equ     type phdr
IHDRLEN         equ     type iphdr
UHDRLEN         equ     type udphdr

PMAXDAT         equ     MAXPKTLEN - PHDRLEN
UMAXDAT         equ     MAXPKTLEN - PHDRLEN - IHDRLEN - UHDRLEN

packet  struc
                db      PHDRLEN dup (?)
                db      IHDRLEN dup (?)
                db      UHDRLEN dup (?)
        data    db      UMAXDAT dup (?)
packet  ends

udpconn struc
        fhost   dd      ?
        fport   dw      ?
        lport   dw      ?
udpconn ends

CONNLEN         equ     type udpconn

IHDR            equ     PHDRLEN
UHDR            equ     IHDR + IHDRLEN
```

```
;mac.inc - some useful macros

pushall macro
                push ax
                push bx
                push cx
                push dx
                push es
                push ds
                push si
                push di
                push bp
                pushf
        endm

popall  macro
                popf
                pop bp
                pop di
                pop si
                pop ds
                pop es
                pop dx
                pop cx
                pop bx
                pop ax
        endm

moveit  macro
                local   iseven
                shr     cx,1
                jnc     iseven
                movsb
iseven:         rep     movsw
        endm

stk_switch      macro   tos
                mov     word ptr old_stk + 2,ss
                mov     word ptr old_stk,sp
                mov     ax,ds
                mov     ss,ax
                mov     sp,offset DGROUP:tos
                endm

stk_restore     macro
                mov     ss,word ptr old_stk + 2
                mov     sp,word ptr old_stk
                endm

old_stk_ptr     macro
old_stk         dd      ?
                endm
```

```
/*      buf.c - Buffer management routines      */

#include "buf.h"

static BUFFER bufpool[NBUFS];           /* the actual buffer pool */

PACKET *allocp (void)           /* return a pointer to a free packet buffer */
{
        register BUFFER *b;

        for (b = bufpool; b < &bufpool[NBUFS]; b++)
                if (b->status == B_FREE)
                {       b->status = B_RSVD;
                        return (PACKET *) b;
                }

        return NULL;
}

void freep (PACKET *p)          /* free the packet buffer allocated by allocp */
{
        if (p != NULL)
                ((BUFFER *) p)->status = B_FREE;
}
```

```
/*      q.c - Queue management routines */

#include "q.h"
#include <dos.h>


void enque (register Q *q, PACKET *p)   /* adds p to the tail of q */
{
        register BUFFER *b = (BUFFER *) p;

        if (q == NULL)
                return;

        if (isempty (q))
                q->head = b;
        else
                q->tail->next = b;

        (q->tail = b)->next = NULL;
}


PACKET *deque (register Q *q)
/* removes the packet at the head of the queue and returns a pointer to it */
{
        register BUFFER *b;

        if (q == NULL || isempty (q))
                return NULL;
```

```
        disable();
        q->head = (b = q->head)->next;
        enable();

        return (PACKET *) b;
}
```

```
/*      port.c - port management routines       */

#include "port.h"

static PORT port[NPORTS];

int psend (unsigned pno, PACKET *p)
/* send a packet to a port, returns 0 on success */
{
        if (!pvalid (pno))
                return -1;

        enque (&port[pno].q, p);
        return 0;
}


PACKET *precv (unsigned pno)
        /* read a packet from a port
         * returns a pointer to it or NULL if no packets are pending
         * tries NTRIES times before giving up */

{
        register Q *q;
        register PACKET *p;
        unsigned long i;

        if (!pvalid (pno))
                return NULL;

        if ((p = deque (q = &port[pno].q)) != NULL)
                return p;

        for (i = NTRIES; isempty (q) && i; i--)
                ;

        return (i? deque (q) : NULL);
}


int popen (unsigned n)
        /* open a port, returns port no. or -1 for error
         * n = port no. desired or ANYPORT if any port acceptable */
{
        register int i;
        register PORT *p;
```

```
        if (n < NPORTS  &&  port[n].status == P_FREE)
        {       port[n].status = P_RSVD;
                return n;
        }

        else if (n == ANYPORT)
        {       for (i = 0, p = port; i < NPORTS; i++, p++)
                        if (p->status == P_FREE)
                        {       p->status = P_RSVD;
                                return i;
                        }
        }

        return -1;
}


void pclose (unsigned pno)
        /* close a previously opened port */
{
        if (pvalid (pno))
        {       pclear (pno);
                port[pno].status = P_FREE;
        }
}


int pstat (unsigned pno)
        /* port status - invalid : -1, empty queue : 0, else : 1 */
{
        return ( (!pvalid (pno))? -1
                : isempty (&port[pno].q)? 0
                : 1 );
}


void pclear (unsigned pno)
        /* clear all packets pending at a port */
{
        register Q *q;
        register PACKET *p;

        if (!pvalid (pno) || isempty ((q = &port[pno].q)))
                return;

        while ((p = deque (q)) != NULL)         /* remove each packet */
                freep (p);                      /* and free its buffer */
}
```

```
/* udp.c - UDP datagram sending and demultiplexing routines */

#include "udp.h"
#include "port.h"
```

```
        if (n < NPORTS  &&  port[n].status == P_FREE)
        {       port[n].status = P_RSVD;
                return n;
        }

        else if (n == ANYPORT)
        {       for (i = 0, p = port; i < NPORTS; i++, p++)
                        if (p->status == P_FREE)
                        {       p->status = P_RSVD;
                                return i;
                        }
        }

        return -1;
}


void pclose (unsigned pno)
        /* close a previously opened port */
{
        if (pvalid (pno))
        {       pclear (pno);
                port[pno].status = P_FREE;
        }
}


int pstat (unsigned pno)
        /* port status - invalid : -1, empty queue : 0, else : 1 */
{
        return ( (!pvalid (pno))? -1
                : isempty (&port[pno].q)? 0
                : 1 );
}


void pclear (unsigned pno)
        /* clear all packets pending at a port */
{
        register Q *q;
        register PACKET *p;

        if (!pvalid (pno) || isempty ((q = &port[pno].q)))
                return;

        while ((p = deque (q)) != NULL)         /* remove each packet */
                freep (p);                      /* and free its buffer */
}
```

```
/* udp.c - UDP datagram sending and demultiplexing routines */

#include "udp.h"
#include "port.h"
```

```
#include "netut.h"

static struct phdr phdr = {0, 0, 0, UDP_PROT, 0};       /* the pseudo-header */

int udpsend (UDPCONN *u, PACKET *p, int datlen)
{
        register struct udp *udpp = UDPPTR (p);
        register int udplen = UHDRLEN + datlen;

        IPPTR (p)->hdr.prot = UDP_PROT;
        udpp->hdr.sport = bswap (u->lport);
        udpp->hdr.dport = bswap (u->fport);

        if (udplen & 0x1)
                ((char *) udpp)[udplen] = '\0';

        phdr.len = udpp->hdr.len = bswap (udplen);

/*      code to be included if checksum is to be calculated :
 *      phdr.src = myaddr;
 *      phdr.dest = lswap (u->fhost);
 *
 *      udpp->hdr.chksum = chksum ((int *)&phdr, sizeof (struct phdr) >> 1);
 *      udpp->hdr.chksum = ~chksum ((int *)udpp, (udplen + 1) >> 1);
 */
        udpp->hdr.chksum = 0;

        return (ipsend (u->fhost, p, udplen));

}


int udpdemux (PACKET *p, int len)
{

        register struct udp *udpp = UDPPTR (p);
        unsigned xsum;

        if (len != bswap (udpp->hdr.len))
                return 1;

        if ((xsum = udpp->hdr.chksum) != 0)
        {       if (len & 0x1)
                        ((char *) udpp)[len] = '\0';
                phdr.src = IPPTR (p)->hdr.src;
                phdr.dest = IPPTR (p)->hdr.dest;
                phdr.len = udpp->hdr.len;

                udpp->hdr.chksum = chksum ((int *)&phdr, sizeof (struct phdr) >> 1);
                if ((udpp->hdr.chksum = ~chksum ((int *)udpp, (len + 1) >> 1)) != xsum)
                        return 1;
        }

        return (psend (bswap (udpp->hdr.dport), p));    /* enque at the specified port */
}
```

```
/* ip.c - IP datagram sending and demultiplexing routines */

#include "ip.h"
#include "udp.h"
#include "net.h"
#include "buf.h"
#include "netut.h"
#include "strtoip.h"

static unsigned int ipid = 0;           /* the datagram sequence number */
IPADDR myaddr;                          /* local internet address (network byte order) */
IPADDR mynet;                           /* network internet address (host byte order) */

int ipsend (IPADDR fhost, register PACKET *p, int datlen)
{
        register struct ip *ipp = IPPTR (p);

        p->hdr.type = IP;
        ipp->hdr.ver_hlen = IVER_HLEN;
        ipp->hdr.srvctyp = ISRVCTYP;
        ipp->hdr.len = bswap (IHDRLEN + datlen);
        ipp->hdr.id = bswap (ipid++);
        ipp->hdr.frag = IFRAG;
        ipp->hdr.ttl = ITTL;
        ipp->hdr.chksum = 0;
        ipp->hdr.src = myaddr;
        ipp->hdr.dest = lswap (fhost);
        ipp->hdr.chksum = ~chksum ((int *)ipp, IHDRLEN >> 1);

        return (send (fhost, p, IHDRLEN + datlen));
}


int ipdemux (PACKET *p, int len)
{
        register struct ip *ipp = IPPTR (p);
        register unsigned xsum;

        if (len != bswap (ipp->hdr.len) || ipp->hdr.ver_hlen != IVER_HLEN)
                return 1;

        xsum = ipp->hdr.chksum;
        ipp->hdr.chksum = 0;
        if ((ipp->hdr.chksum = ~chksum ((int *)ipp, IHDRLEN >> 1)) != xsum)
                return 1;

        if (ipp->hdr.frag != 0)
                return 1;

        switch (ipp->hdr.prot)
        {
                case UDP_PROT :
                        return (udpdemux (p, len - IHDRLEN));
```

```
                default :
                        return 1;
        }
}


int setaddr (char far *s)
{
        if ((myaddr = strtoip (s)) == 0)
                return 1;

        mynet = getnet (lswap (myaddr));
        return 0;
}
```

```
/*      net.c */

#include "ip.h"
#include "netut.h"

int netout (PACKET *p);
extern IPADDR mynet;

/* send a complete IP datagram as a PC-LAN frame */
int send (IPADDR fhost, PACKET *p, int len)
{
        if (len > PMAXDAT)
                return 1;
                                                /* fill in the PC-LAN header. */
        p->hdr.dest = (fhost == mynet)? MYBCAST
                    :(getnet (fhost) == mynet)? getnode (fhost)
                    :MYGATEWAY;
        p->hdr.len = bswap (len);

        return (netout (p));                    /* write to the IMP card */
}


/* PC-LAN packet demuliplexing - called by netin right after a complete packet has been received */
int demux (register PACKET *p)
{
        register int len;

        if ((len = bswap (p->hdr.len)) == 0)
                return 1;

        switch (p->hdr.type)
        {
                case IP :
                        return (ipdemux (p, len));
                default :
                        return 1;
        }
}
```

```
/* strtoip.c - routine for converting strings like "192.0.0.4" to internet addresses in network byte order */


long strtoip (char far *s)
{
        union
        {       long l;
                unsigned char c[4];
        }u;

        unsigned char c, *a = u.c;
        int i = 3, n = 0;

        for (;;)
        {       switch (c = *s++)
                {       case '1' :
                        case '2' :
                        case '3' :
                        case '4' :
                        case '5' :
                        case '6' :
                        case '7' :
                        case '8' :
                        case '9' :
                        case '0' :      n = 10 * n  +  c - '0';
                                        break;

                        case '.' :      if (n > 255 || i-- == 0)
                                                return 0L;
                                        *a++ = n;
                                        n = 0;
                                        break;

                        default  :      if (n > 255 || i != 0)
                                                return 0L;
                                        *a = n;
                                        return u.l;
                }
        }
}
```

```
;pclan.asm - PC-LAN interface routines

__TINY__        equ     1

include rules.asi
include pclan.inc
include mac.inc

@wait   macro   bit                             ;wait till status bit set
        local   wwt
        mov     dx,bx
```

```
wwt:    in      al,dx
        and     al,&bit
        jz      wwt
        endm

@netout macro   port,byt
        mov     dx,&port
        mov     al,byte ptr &byt
        out     dx,al
        endm


LANDATA equ     340h                    ;data i/o port
LANCSR  equ     348h                    ;control and status register

;       ------------------      LANCSR bits ------------------------

TXBF    equ     80h                     ;transmit buffer full
EXPB    equ     20h                     ;expecting a byte
IBE     equ     10h                     ;input buffer empty
SPEC    equ     40h                     ;special byte from card
NEW     equ     01h                     ;new packet
PCMD    equ     02h                     ;treat the next bytes as commands
RTS     equ     01h                     ;request to send

;       ---------------------------------------------------

params  struc
                dw      ?               ; bp
                dw      ?               ; ip
        par1    dw      ?
params  ends

par     equ     [bp]

Header@

CSeg@

_txrdy          proc    near            ;int txrdy (void)
                                        ;returns 0 if ready
        mov     dx,LANCSR
        in      al,dx
        and     al,TXBF
        ret

_txrdy          endp

_netout proc    near            ;int netout (void *packet)
                                ;write a complete packet to the card,returns 0 for success
        push    bp
        mov     bp,sp
        push    si
        push    di
```

```
        mov     bx,LANCSR
        mov     di,LANDATA

        mov     dx,bx
        mov     si,0ffffh
w0:     in      al,dx
        and     al,TXBF
        jz      w1
        dec     si
        jnz     w0

        mov     ax,1
        jmp     short abort             ;transmit buffers full - abort

w1:     @netout bx,PCMD

        @wait   IBE
        @netout di,RTS

        @wait   IBE
        @netout bx,0

        @wait   EXPB
        mov     si,par.par1
        cld

        @wait   IBE
        lodsb                           ;dest
        mov     dx,di
        out     dx,al

        @wait   IBE
        inc     si                      ;ctrl
        inc     si                      ;src
        lodsb                           ;type
        mov     dx,di
        out     dx,al

        @wait   IBE
        inc     si                      ;len (hi)
        lodsb                           ;len (low)
        mov     dx,di
        out     dx,al

        mov     cl,al
        xor     ch,ch
        jcxz    exit                    ; zero length packet

w2:     @wait   IBE
        lodsb
        mov     dx,di
        out     dx,al                   ;write data bytes
        loop    w2
```

```
exit :  xor     ax,ax


abort:
        pop     di
        pop     si
        pop     bp
        ret

_netout endp


_net_in         proc    far
;ISR for packet reception from the card

        push    ax
        push    ds
        mov     ax,cs
        mov     ds,ax
        push    bx
        push    dx

        mov     dx,LANCSR
        in      al,dx
        and     al,SPEC
        jz      ord

        mov     dx,LANDATA
        in      al,dx
        cmp     al,NEW                  ;check if new packet
        je      ni0
        jmp     near ptr endit

ni0:    mov     ax,rbuf
        or      ax,ax
        jnz     ni1
        push    cx
        push    es
        call    near ptr _allocp        ;allocate a packet buffer
        pop     es
        pop     cx
        or      ax,ax
        jz      endit
        mov     rbuf,ax

ni1:    mov     p,ax

        mov     cnt,PHDRLEN
        mov     ishdr,1                 ;reading in the header
        jmp     short endit

ord :
        mov     dx,LANDATA
        in      al,dx                   ;get byte from card
        mov     bx,p
```

```
        or      bx,bx
        jz      endit
        mov     [bx],al                 ;store byte
        inc     p
        dec     cnt
        jnz     endit

        cmp     ishdr,0
        je      over

        dec     bx
        mov     ax,[bx]
        xchg    ah,al                   ;get length of packet from header
        mov     cnt,ax
        mov     ishdr,0                 ;now reading in the data
        jmp     short endit

over :
        stk_switch int_tos

        push    cx
        push    es
        push    rbuf
        call    near ptr _demux         ;try to send the packet to where it ought to go
        or      ax,ax
        jz      ni2
        call    near ptr _freep         ;nobody wants it - I don't either; so free it

ni2:    inc     sp
        inc     sp
        pop     es
        pop     cx

        stk_restore

        mov     rbuf,0
        mov     p,0

endit : cli
        mov     al,20h                  ; EOI to the 8259A
        out     20h,al

        pop     dx
        pop     bx
        pop     ds
        pop     ax
        iret

_net_in         endp

CSegEnd@

DSeg@

rbuf            dw      0
```

```
dnt             dw      0

ishdr           db      1

old_stk_ptr                     ;space to save the original ss and sp when stack-switching

DSegEnd@

        extrn   int_tos : word
        extrn   _allocp : near
        extrn   _freep : near
        extrn   _demux : near

        public  _txrdy
        public  _netout
        public  _net_in

        end
```

```
;netdrvr.asm - the network device driver

include pclan.inc
include mac.inc

;status bits defined in req. hdr. status word

ERROR           equ     8000h
DONE            equ     0100h
BUSY            equ     0200h

;error codes defined for req. hdr. status word (error bit already set)

ILLCMD          equ     0003h or ERROR or DONE
NOTRDY          equ     0002h or ERROR or DONE
OK              equ     DONE

req             equ     es:[bx]

reqhdr          struc
                db      ?                       ;length
                db      ?                       ;unit no.
cmd             db      ?                       ;command code
status          dw      ?                       ;status word
                db      8 dup (?)               ;reserved
reqhdr          ends

inithdr         struc
                db      (type reqhdr) dup (?)
units           db      ?
end_ofst        dw      ?                       ;end address
end_seg         dw      ?
cmd_line        dd      ?
inithdr         ends
```

```
rwhdr           struc
                db      (type reqhdr) dup (?)
                db      ?
trf_ofst        dw      ?       ;transfer address offset
trf_seg         dw      ?       ;transfer address segment
cnt             dw      ?       ;no. of bytes
rwhdr           ends

trf             equ     dword ptr trf_ofst

cblk            struc                   ;control block (see dev.h)
                db      (type udpconn) dup (?)
ack             dw      ?
xbuf            dd      ?
rbuf            dd      ?
cblk            ends

_TEXT   segment byte public 'CODE'
_TEXT   ends
_DATA   segment para public 'DATA'
_DATA   ends
_BSS    segment word public 'BSS'
bdataa  label byte
_BSS    ends
_BSSEND segment word public 'BSSEND'
_BSSEND ends

DGROUP  group   _TEXT,_DATA,_BSS,_BSSEND

ASSUME cs:_TEXT, ds:DGROUP

_TEXT segment

                org     0
;header

                dw      -1,-1                   ;pointer to next driver
                dw      0c800h                  ;attribute (CHR, IOCTL, OCRM)
                dw      strategy
                dw      interrupt
                db      'NET     '              ;device name

_TEXT ends

_DATA segment

;call table

cmd_tab label   word
                dw      init
                dw      ill_cmd
                dw      ill_cmd
                dw      ioctl_read
                dw      read
                dw      in_status
```

```
                dw      ill_cmd
                dw      ill_cmd
                dw      write
                dw      write
                dw      out_status
                dw      ill_cmd
                dw      ioctl_write
                dw      open
                dw      close


isopen  db      0                       ; = 0 only when device is not open
ack_due dw      0

old_stk_ptr
reqhdr_ptr      dd      ?

addr_msg        db      'Invalid internet address$'
ver_msg         db      'Incorrect DOS version - use 3.0 or higher$'

_DATA ends


_BSS segment

buffer  packet                          ;transmit buffer
conn    udpconn

old_vector      dd      ?               ; original irq2 vector

_BSS ends


_TEXT segment

strategy        proc    far

                mov     cs:word ptr reqhdr_ptr,bx
                mov     cs:word ptr reqhdr_ptr + 2,es
                ret

strategy        endp


interrupt       proc    far

                pushall
                mov     ax,cs
                mov     ds,ax
                stk_switch      main_tos
                sti
                cld

                les     bx,reqhdr_ptr
                mov     al,req.cmd
```

```
                cmp     al,0eh
                ja      cmd_error

                cbw
                shl     ax,1
                mov     si,ax
                call    cmd_tab[si]

exit:           les     bx,reqhdr_ptr
                mov     req.status,ax
                stk_restore
                popall
                ret

cmd_error:
                mov     ax,ILLCMD
                jmp     short exit

interrupt       endp


ill_cmd         proc    near

                mov     ax,ILLCMD
                ret

ill_cmd         endp


write           proc    near

                mov     cx,req.cnt              ;check length of data
                cmp     cx,UMAXDAT
                jbe     iw1
                mov     cx,UMAXDAT
                mov     req.cnt,cx

iw1 :           mov     bp,cx
                mov     dx,ds

                lds     si,req.trf
                mov     es,dx
                mov     di,offset DGROUP:conn
                mov     cx,CONNLEN
                moveit                          ;get connection info

                lodsw
                mov     cs:ack_due,ax           ;ack flag

                lds     si,[si]
                mov     di,offset DGROUP:buffer.data
                mov     cx,bp
                moveit                          ;get data to be sent

                mov     ds,dx
```

```
                push    bp
                mov     ax,offset DGROUP:buffer
                push    ax
                mov     ax,offset DGROUP:conn
                push    ax
                call    near ptr _udpsend       ;send it
                add     sp,6

                or      ax,ax
                jnz     not_rdy
                cmp     ack_due,0               ;wait for reply if ack flag set
                je      ok

                push    conn.lport
                call    near ptr _pclear        ;clear all waiting messages,
                inc     sp                      ;we're interested in the reply to this one
                inc     sp
                les     bx,reqhdr_ptr
                call    near ptr _read
                jcxz    not_rdy
                les     bx,reqhdr_ptr
                mov     req.cnt,cx

ok:             mov     ax,OK
                ret

not_rdy :       mov     ax,NOTRDY
                ret

write           endp


out_status      proc    near

                call    near ptr _txrdy
                jnz     os1
                mov     ax,OK
                ret

os1 :           mov     ax,BUSY or DONE
                ret

out_status      endp


_read   proc    near

                les     di,req.trf
                push    es:[di].lport
                call    near ptr _precv         ;read port
                inc     sp
                inc     sp
                or      ax,ax
                jz      no_pkts
```

```
                mov     si,ax
                mov     dx,ax

                mov     ax,[si][UHDR].udp_len           ;get length of data
                xchg    ah,al
                sub     ax,UHDRLEN
                cmp     ax,UMAXDAT
                jbe     _r0
                mov     ax,UMAXDAT
_r0:            mov     cx,ax

                les     bx,reqhdr_ptr
                les     di,req.trf
                mov     ax,word ptr [si][IHDR].ip_src + 2       ;get connection info
                xchg    ah,al
                stosw
                mov     ax,word ptr [si][IHDR].ip_src
                xchg    ah,al
                stosw
                mov     ax,[si][UHDR].udp_sport
                xchg    ah,al
                stosw

                lea     si,[si].data
                les     di,es:[di][8]
                mov     bp,cx
                moveit                          ;transfer data to caller's buffer

                push    dx
                call    near ptr _freep         ;free the buffer holding the packet
                inc     sp
                inc     sp

                mov     cx,bp
                ret

no_pkts :       xor     cx,cx
                ret

_read   endp


read            proc    near

                call    near ptr _read
                les     bx,reqhdr_ptr
                mov     req.cnt,cx
                mov     ax,OK
                ret

read            endp
```

```
in_status       proc    near

                mov     ax,OK
                ret

in_status       endp


open            proc    near

                cmp     isopen,0
                jne     op0

                xor     bx,bx
                mov     es,bx
                mov     bx,0ah * 4
                cli
                mov     ax,es:[bx]                      ;save and change irq2 vector
                mov     word ptr old_vector,ax
                mov     word ptr es:[bx],offset DGROUP:_net_in
                inc     bx
                inc     bx
                mov     ax,es:[bx]
                mov     word ptr old_vector + 2,ax
                mov     es:[bx],cs
                sti

                xor     ax,ax
                out     21h,al

op0:            inc     isopen
                mov     ax,OK
                ret

open            endp

ioctl_read      proc    near
        ;port open call
                les     di,req.trf
                push    es:[di].lport
                mov     si,es
                call    near ptr _popen
                inc     sp
                inc     sp
                mov     es,si
                mov     es:[di].lport,ax
                mov     ax,OK
                ret

ioctl_read      endp


ioctl_write     proc    near
        ;port close call
                les     di,req.trf
```

```
                push    es:[di].lport
                call    near ptr _pclose
                inc     sp
                inc     sp
                mov     ax,OK
                ret

ioctl_write     endp


close           proc    near

                dec     isopen
                cmp     isopen,0
                jne     cl0

                mov     al,4
                out     21h,al

                xor     bx,bx
                mov     es,bx
                mov     bx,0ah * 4
                cli
                mov     ax,word ptr old_vector
                mov     es:[bx],ax
                mov     ax,word ptr old_vector + 2
                mov     es:[bx + 2],ax
                sti

cl0:            mov     ax,OK
                ret

close           endp


init            proc    near

                mov     req.end_ofst,offset DGROUP:endadr@
                mov     req.end_seg,cs

                mov     ax,ds
                mov     es,ax
                xor     ax,ax
                mov     di,offset DGROUP:bdata@
                mov     cx,offset DGROUP:edata@
                sub     cx,di
        rep     stosb                           ; initialize variables in the _BSS segment

                mov     ah,30h
                int     21h                     ;check DOS version
                cmp     al,3
                jae     i0
                mov     dx,offset DGROUP:ver_msg
                jmp     short abort
```

```
i0:             les     bx,reqhdr_ptr
                les     di,req.cmd_line
                mov     cx, 0ffffh
                mov     al, ' '
                repe    scasb
                repne   scasb
                dec     di
                repe    scasb
                dec     di
                push    es
                push    di
                call    near ptr _setaddr       ;get internet address string from command line
                add     sp,4
                or      ax,ax
                jz      i1
                mov     dx,offset DGROUP:addr_msg

abort:          les     bx,reqhdr_ptr           ;cannot be installed
                mov     req.end_ofst,0
                mov     req.units,0
                mov     ah,9
                int     21h

i1:             mov     ax,OK
                ret

init            endp

_TEXT ends

_BSSEND segment
edata@  label   byte

even
        db      32 dup ('MSTACK  ')             ;stack for main routines
main_tos        label   word
        db      32 dup ('ISTACK  ')             ;stack for the ISR netin
int_tos         label   word
        db      32 dup (?)

endadr@ label   byte
_BSSEND ends

extrn   _txrdy : near
extrn   _udpsend : near
extrn   _netout : near
extrn   _net_in : near
extrn   _popen : near
extrn   _pclose : near
extrn   _pclear : near
extrn   _precv : near
extrn   _freep : near
extrn   _setaddr:near
public int_tos
end
```

```
; netut.asm - network utility functions

__TINY__        equ     1

include rules.asi

Header@

CSeg@

_bswap          proc    near            ;int bswap (int)
                                        ; swap bytes in a word
        mov     bx,bp
        mov     bp,sp

        mov     ax,[bp][2]
        xchg    ah,al

        mov     bp,bx
        ret

_bswap          endp

_lswap          proc    near            ;long lswap (long)
                                        ;swap bytes in a long
        mov     bx,bp
        mov     bp,sp

        mov     ax,[bp][2]
        mov     dx,[bp][4]
        xchg    dh,al
        xchg    ah,dl

        mov     bp,bx
        ret

_lswap          endp


_chksum         proc    near            ;int chksum (int *p, int n)

        mov     bx,bp
        mov     bp,sp
        mov     es,si

        mov     si,[bp][2]
        mov     cx,[bp][4]
        xor     dx,dx

cs0:    lodsw
        adc     dx,ax
        loop    cs0
```

```
        adc     dx,0
        mov     ax,dx

        mov     si,es
        mov     bp,bx
        ret

_chksum         endp

CSegEnd@

public  _bswap
public  _lswap
public  _chksum

end
```

REFERENCES

[1] Low-cost implementation of PC-LAN, Sanjeev Verma, M.Tech. thesis, January 1988, EE Department, IIT Kanpur.

[2] A VME-bus interface for a low-cost LAN, N.R. Gogate, M.Tech. thesis, January 1989, EE Department, IIT Kanpur.

[3] Operating Systems Design, Vol.II: Internetworking with XINU, Douglas Comer, Prentice Hall, 1986.

[4] Computer Networks, Andrew S. Tanenbaum, Prentice Hall of India, 1985.

[5] MS-DOS Developer's Guide, John Angermeyer and Kevin Jaeger, Howard W. Sams & Co., 1986.

[6] Advanced MS-DOS, Ray Duncan, Microsoft Press.

[7] DOS Technical Reference